ADVANCED DECISION SYSTEMS

THE BATTLEFIELD COMMANDER'S ASSISTANT PROJECT: RESEARCH IN TERRAIN REASONING

ADS TR-1058-01

May 22, 1987

FINAL REPORT

for the Period 16 January 1984 - 1 April 1987

Contract No: DAAB07-84-C-K516

ADS Project No: 1058

Prepared by:

Daniel G. Shapiro

Carl Tollander

Advanced Decision Systems

201 San Antonio Circle, Suite 286

Mountain View, CA 94040

Approved for public release; distribution is unlimited.

Prepared for:

Center For Communications/Automatic Data Processing

Headquarters, Army Communications and Electronics Command

Ft. Monmouth, New Jersey 07703

201 San Antonio Circle, Suite 286
Mountain View, California 94040-1289
415/941-3912 FAX: 415/949-4029

Table of Contents

List of Figures

List of Tables

Acknowledgements:

As the project leader, I would like to thank all of the individuals who contributed to the BCA project during its development. In particular, Carl Tollander performed all implementation of the Multiple Pane Interface, conducted all systems integration activities. and provided major design support. John Woodfill designed and implemented the terrain query language, which Victor Askman later expanded. Randal Walser provided background for determining the desired I/O of our terrain analysis prototype by examining military scenarios. Jeffrey Abram and Philip Marks were involved with the early phases of the project where we identified terrain reasoning as the appropriate application area, and implemented a first terrain manipulation and display prototype.

1. Introduction

This document is the final report for the Battlefield Commander's Assistant project (BCA), which was a three year, three man-year effort to first identify, and then prototype an artificial intelligence based tool for supporting battalion operations. After a certain amount of exploratory efforts, we selected terrain reasoning as the most important (high impact) application area, and ultimately created a terrain analysis and planning system (TAPS) prototype, which has been delivered to the sponsors.

This report documents TAPS in its entirety, but for completeness we also discuss our early efforts in identifying potential application areas, and in exploring alternative approaches to some of the technical problems of terrain reasoning. A number of these ideas are worth pursuing in the future, even if they were not incorporated into TAPS.

Partial contents: (To P.4)

The outline of this report is as follows: Chapter 2 describes our efforts towards identifying high impact application areas, Chapter 3 defines the terrain reasoning problem we ultimately adopted, Chapter 4 describes some of our technical explorations into that topic, Chapter 5 documents TAPS, and Chapter 6 discusses future directions. We have added several appendices: appendix I lists documents examined in the BCA literature search, appendices II and III provide 35 millimeter slides of the SMAP and TAPS system outputs respectively (note that only 3 copies of the report have been delivered with this material), and appendix IV gives instructions for loading and running the TAPS system.

As a compendium of our work, much of the material in this report was obtained by abstracting sections from earlier project technical reports. The interested reader is referred to [Shapiro 85a], [Shapiro 85b], and [Shapiro 86] for details.

2. Domain Survey

The first phase of the BCA project was concerned with identifying application areas for AI technology in battalion operations. To give a sense of scope, our starting point was the abstract system design shown in figure 2-1, which covers plan generation, evaluation and monitoring in its entirety, as well as issues of situation assessment, data acquisition, and order dissemination.

In order to restrict the survey problem to a manageable level, we made the a priori decision to focus on activities related to combat planning and then examined battalion operations in that area. We were looking for potential applications which were

- important,
- difficult or time consuming to solve by present means, and
- promising for solution by computer.

Here, an "important" problem is critical to mission success, readiness or the safety of the battalion, while a "difficult" problem is one in which unsatisfactory or suboptimal solutions are often arrived at even by competent staff.

The details of our study are a bit too lengthy to describe here, since it involved a significant amount of interaction with staff officers at corps through battalion levels, and caused us to examine a collection of documents about battalion operations, existing automation, and military theory. (See [Shapiro 85a], and appendix I.) The conclusions, however, are easy to summarize; it turns out that the response times and resource constraints applicable at battalion level naturally emphasize terrain concerns above situation assessment (including enemy intention analysis), logistics, complex plan generation and execution monitoring. As a result, we selected terrain reasoning as the critical problem to pursue, and ultimately developed the Terrain Analysis and Planning System (TAPS) which is described later in this report.

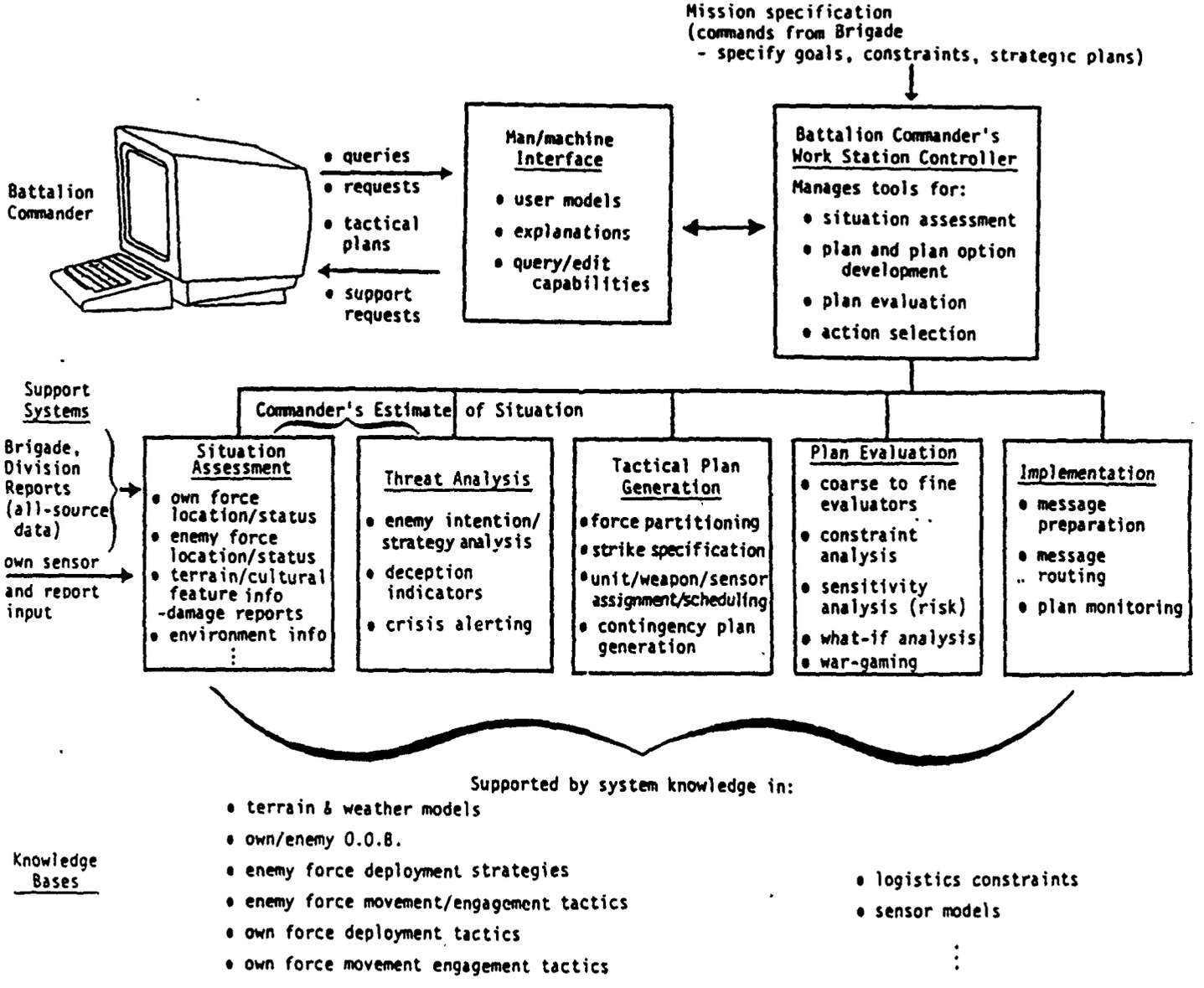


Figure 2-1: An abstract design for a Battalion Commander's Assistant

This understanding of battalion level concerns was a valuable outcome of our domain survey, and it is worth discussing in more detail. The following section tries to illustrate the change in emphasis between battalion and corps level combat planning, and the types of AI applications which are therefor most critical to each.

2.1 Potential application areas

It is not difficult to identify potential applications of AI in each of the areas governed by the principal staff officers; intelligence, operations and logistics. (We omit the personnel and political liaison roles as they are less immediately involved in combat decision making.) With respect to intell, situation assessment in the broad sense is clearly important; this includes identifying enemy units, their status and positions from visual and electronic observations, and determining their intent from these particulars given knowledge of the battle. The central function of operations is to create plans and monitor their execution, both of which are receiving attention as AI applications at the Corps and Division level (see [Stachnick 87], [Payne 86], and [Loberg 86a, Loberg 86b]). In logistics, the problem of course is to apportion, store and assure delivery of resources such that they match anticipated demand. Given the extremely resource intensive nature of military operations, logistics discussions can in fact precede battle planning.

The situation assessment, planning and logistics problems occur at all echelon, but their magnitude changes drastically with size. At Corps, the battlefield typically occupies thousands of square kilometers, commands 50,000 men with thousands of vehicles, and the plans govern actions for many days at a time. In contrast, a battalion battlefield is less than 50 square kilometers, with $\sim$500 men and a 6 to 12 hour planning/execution cycle time. Each of these scaling factors alters the nature of the intell, operations and logistics tasks, and therefor the type of computer support that might be required. The following paragraphs expand on this statement by considering each potential application area in turn. Some of the observations we provide have been abstracted from our work on the Air Land Battle Management Study ([Stachnick 87]) which was tasked to compare AI planning techniques with the requirements of corps level planning.

As a consequence of battlefield size, corps level intell requires deep looking sensors, many different types of sensors, and a large number of observations to gain a picture of the current situation. There are many enemy units with a wide choice of objectives, implying that rather global inference is necessary in order to project their intent. From a technical perspective, these problems translate into complex hypothesis formation, and multisensor integration tasks. Both of these will benefit from AI based automation.

In contrast, the restricted battlefield size at battalion implies a reduced need for sensor resources in each of the dimensions of type, quantity and depth-of-field. In fact, battalions typically operate on direct visual observation (which is not to say that they won't be able to make use of more indirect sensors in the future). This implies that multisensor integration is clearly not important, while the hypothesis formation problem remains, but is greatly reduced in scope. Our domain study concluded however, that enemy positions and likely objectives can be inferred directly from the characteristics of the terrain given a rough knowledge of their mobility and weaponry. This was a major piece of support for our focus on terrain reasoning.

The quantity and variety of the forces at corps suggest that the planning problem is more complicated, although that issue is debatable; there are certainly more resources involved and therefor more options for employing/allocating them (and attendant coordination details), but the military command structure keeps the number of units directly tasked roughly equivalent (e.g., three divisions to a corps, three companies to a battalion). In addition, a maxim of military operations is that plans must be simple in order to be executed successfully. As a result, we can expect the complexity of an operations concept to be low, presumably regardless of echelon.

The time scale of the plans, however, has a drastic effect on their nature and on the type of automated support which should be supplied. At corps, the size of the battlefield and mobility of units is such that situations can only change at a moderate rate. As a result, execution of plans at this level (i.e., *operations* has a certain minimum duration which is measured in days, and it only makes sense to consider

replanning on intervals of that order. In this environment, automated planning support systems can clearly take on the order of hours to execute.

In addition, there are timelines to be respected in both the gathering and processing of intelligence information, and in the planning time to be allocated to subordinate units once the corps has completed its decision cycle. These effects imply that corps level plans are built on information which may be up to several days old (by the time execution begins), and that they must be abstract enough to accommodate change in the situation before they are enacted, and of course during the period while they are in force. As a result, corps level plans often read more like a refinement of problems for subordinate units to solve than a collection of specific actions to perform. It follows that automated tools which support execution monitoring and plan refinement are critical at higher echelon (since the corps needs to detect and react to situation changes). However, the technology of AI planning has not explored the concept of problem refinement (vs. action selection) to any degree. This implies that applications in plan generation and plan refinement have technical obstacles to surmount.

In contrast, battalion level planning has a very immediate nature. Situation data is fresh, the planning itself takes a small amount of time, and the plans govern a period that projects only small distance into the future (typically 6 to 12 hours). As a result, the plans tend to be very specific in terms of objectives and intent, and their success/failure is directly observable (unlike the process of monitoring at corps, which requires projection further into an uncertain future). The implication here is that the timeline is short enough that all planning tools will need to be quite fast, and that the role of monitoring and plan refinement is more limited at battalion than corps.

Our domain study also observed that terrain considerations placed a great deal of structure on battalion level plans, to the point of identifying specific tactical objectives and subobjectives, as in the selection of key terrain within an avenue of approach, or in the choice of river crossing sites, which requires selecting covering force sites and estimations of enemy observation areas. In fact, in the words of one battalion

commander we consulted, "if you know the enemy and your own capabilities, one look at the terrain and the right course of action is obvious". (This greatly emphasized the implementation of terrain reasoning tools before planning aides in our minds.) The underlying reason for this specificity of terrain on planning at battalion level may be because the areas of operation are not significantly larger than the predominant weapons ranges; local terrain considerations therefor effect every decision of force employment.

The potential for AI applications to battalion logistics is more simple to analyze. At corps, the quantities of equipment and resources directly imply massive logistics concerns. Ability to transport tonnage strongly affects our worldwide military capability (thus the appearance of rapid deployment forces), and once forces are in place, the available stockpile of supplies, their allocation and prioritization in many ways determine the potential for success of any operation. The number of vehicles is also large enough that the effect of the equipment on the terrain also has to be calculated. Some movements are not feasible simply because they would reduce the environment to impassable mud. Viewing corps level logistics as a resource allocation process, automation will need to support demand scheduling with changing priorities, partial satisfaction and interrupts.

At battalion, most of these concerns simply go away. Battalions typically carry several days worth of their own supplies, which means that the complexity of the logistics chain is not at issue. Similarly, because the duration of battalion missions is short, having the logistics base for sustained operations is more a matter for superiors. Battalions also have a much smaller pool of owned resources to impact the force allocation problem. As a result, battalions are much more focused on tactical considerations.

In summary then, while battalion level situation assessment, and course of action generation, evaluation and monitoring are all potential application areas for artificial intelligence, the nature of the battalion level combat problem is extremely focused on

the terrain. As a result, we selected terrain reasoning (above planning) as the critical application area, and developed the TAPS system discussed later in this report. The nature of the terrain reasoning problem is discussed in the following chapter.

3. The Terrain Reasoning Problem

In our conception, the purpose of a military terrain reasoning is to support tactical decision making. From a domain point of view then, the object is to use knowledge of the terrain to answer questions that the intelligence, logistics, or planning staff phrase, or those requests made directly by the unit commander. It is easy to generate an extensive list of such requests. For example;

- Sensor processing:
  - Is a detection of a vehicle or unit consistent with the constraints imposed by the underlying terrain?
  - Where can we expect a vehicle or unit to deploy given our knowledge of the terrain (and unit mobility, or tactical capability)?
- Situation Assessment:
  - What pieces of terrain (including cultural features) constitute likely enemy objectives? Likely targets? Where is the enemy likely to be?
  - Given the tactical situation and the terrain, what is the enemy likely to do next?
- Logistics:
  - What are the main supply routes in a given area?
  - What is the capacity of a road network for transporting units?
- Planning:
  - Where should we choose to fight (what types of terrain are most advantageous to us and least advantageous to the enemy)?
  - What tactics (singly, or in coordination) make best use of the terrain? How should the terrain be enhanced?
  - How much terrain (and which pieces) should be delegated to subordinate units?
  - What are the logical avenues of approach given our knowledge of unit mobility, and what size force can they support?

- In economy of force operations, what is the "terrain multiplier"? How little force can we use to impede an enemy advance down a particular avenue?
- In offensive operations, what are the natural barriers or objectives that define good phase lines?
- How quickly can we expect to move over the terrain, given unit composition and expected resistance?
- How will characteristics of the terrain affect the outcome of a battle?
- How vulnerable is a given route?

• Terrain analysis:

- Which natural features form critical terrain?
- Where should a given weapons system be deployed for maximum effect?
- What terrain supports a particular operation, such as a river crossing, a helicopter landing site, or movement across a front?
- What is the trafficability through a region by vehicle platform?
- How will the terrain change through use or weather?

This list attempts to identify questions that are principally terrain oriented, although it is clear that information about unit, vehicle, and weapons system capabilities is also required. In some cases, doctrinal knowledge about enemy choices is relevant (e.g., in asking where the enemy is likely to go next and with what types of units), as is information about combat modelling (for predicting conflict outcomes) and tactical options and principles. For example, one has to know the tactical concept of shaping battles in order to decide to deploy mines which will deny the enemy access to a particular route. Portions of such questions (where to deploy a minefield *given* the desire to restrict access to an objective) are clearly within the scope of terrain reasoning.

The question then is to ask where terrain reasoning ends, and intelligence

processing, logistics, force allocation, and tactical planning begin. It is clear that all of these applications require information about the terrain, but that each employs a body of specialized knowledge as well. By way of response, the extreme positions are to support operations which only reference the primitive elevation and feature data supplied by digitized terrain databases (with no embedded concept of military units or their capabilities), or to expand terrain reasoning to encompass arbitrary amounts of plan generation or situation assessment specific structures.

By this interpretation, any solution to the terrain reasoning problem will appear ad hoc from some dimension, since the dividing lines are inexact. Our approach has been to identify what appear to be common or important questions, and construct a small set of knowledge bases which are required to answer them. We include as little specialized data as possible.

The design for the TAPS system described in chapter 5 (including both the implemented and unimplemented portions) represents our current view of what that terrain reasoning kernel should contain. In specific, it relies on the following types of knowledge:

- primitive features of the terrain (both natural and cultural) such as roads, rivers, elevation data and land usage,
- the mobility of equipment and military units,
- the capabilities of weapons systems in terms of range, delivery mechanism (direct or indirect) and target types
- the command structure, composition of units, and specific resources involved in a given scenario,
- current knowledge with respect to the order of battle, and
- models for features of tactical interest (which organize the above information around questions of military interest)

Without going too deeply into the algorithms which operate on the above knowledge, it is clear that a range of capabilities is possible. For example, by processing

elevation data we can compute visibility profiles, and fields of fire for specific weapons. Given primitive terrain properties, slope and vehicle data, we can build trafficability maps for vehicles of known types (a terrain analysis activity). Using the ability to determine fields of fire, we can write procedural models which compute good sites for weapons deployment. For example, a direct fire weapon should be at the top of a hill, while certain indirect fire weapons should be out of view from their targets, near the tops of hills to enhance range. This type of information can be used in turn to generate expectations (for the sensor interpretation problem) concerning the probable locations of enemy units. Some other obvious possibilities are to extract natural terrain features from the elevation data (such as ridge lines or hilltops) or to process the local topology (with trafficability) to identify avenues of approach and critical terrain (which has field of fire over large portions of an avenue or avenues).

The ability to write procedural models which manipulate this data provides a powerful capability; it allows a person to express complicated questions about the terrain, and to aggregate component features into a larger whole. We plan to use such models (see section 5.5.2) to define a wide range of military features of interest including river crossing sites (which have substructure composed of fording-points, beach-heads, covering force sites, and potential enemy locations), helicopter landing areas, and potential mine-field empacements. (This vocabulary for military sites is also important as an interface mechanism.) Note that the terrain requirements for all of these features are dependent upon the characteristics of the actual units involved. Even simple feature-extraction questions cannot be answered without reference to these external, non-terrain oriented properties.

We have also considered adding a tactical interpretation of interactions on the battlefield to the terrain reasoning knowledge base. This information would identify relationships of the form, unit A is *attacking* unit B, or that units X, Y and Z are all involved in a particular *thrust* with an attached objective. The presence of this knowledge (once encoded - and selecting the appropriate representation is an open question) would enable a class of analyses which determine the feasibility of actions on

the terrain *given the current combat situation.* For example, we could predict movement rates and anticipated strengths of engaged units (incorporating terrain effects on maneuver and combat) or identify units able to be at a particular place at a particular time. A representation of this kind similarly opens the door to much deeper tactical reasoning, where we explore options for tactical actions suggested by the terrain. At the current time we are not considering this form of data a part of a kernel terrain reasoning system since the knowledge appears to be more specialized to plan generation and evaluation.

It is important to include a word here about what terrain reasoning is *not.* By the definition above, a terrain reasoning kernel does not contain representations or algorithms which are highly application specific. So, for example, the system will not model sequences of actions which compose a military plan, or provide mechanisms for searching among the sets of tactical actions which might accomplish specific goals. The system will, however, aid in generating and evaluating single tactical choices when they are impacted by the terrain (which is often). Similarly, a terrain reasoning kernel will not maintain signal intelligence data, or field surveillance reports which are the underpinning employed by intelligence officers to construct situation assessments. As mentioned earlier, it would be possible for a terrain kernel to aid in sensor interpretation by identifying plausable areas for unit or vehicle deployment. A possible extension is to introduce the concept of *patterns* of deployment (e.g., linear columns or star-shaped radar and missile arrays) into the underlying knowledge base. This would support the sensor interpretation problem, although we currently view the idea as too application specific.

In summary then, the concept behind our view of terrain reasoning is that it identifies (or examines) regions of terrain which have some set of desired properties. These properties may relate in the broad sense to mobility or weapons characteristics, or to the identity and position of specific units, but the end result is ultimately computed by asking the right set of low level questions about the terrain. That is, we compile application questions into requests which constrain the topology or primitive feature

data in the underlying terrain. In all cases, the output is a region, or collection of regions of terrain.

Our prototype for a terrain reasoning system is discussed in chapter 5.

4. Background studies

The following sections describe investigations we performed during the course of the BCA contract which were ultimately not incorporated into TAPS. We discuss several approaches for representing and manipulating physical space, as well as a Symbolic Map Presentation system (SMAP) which gave us exposure to digitized terrain data and suggested many of the architectural ideas in TAPS.

It is important to note that the work on alternate representations was crucial in the development of TAPS, since it led to the conclusion that no single terrain representation was going to be sufficient. As a result, we built TAPS around an abstract query language whose purpose was to hide representational commitments, allowing us to deemphasize the role of any specific technique. As TAPS grows however, it will become important to implement multiple data structures, and the studies described below will have very tangible impact.

4.1 A Comparison of Terrain Data Representations

There are several different ways that terrain data can be represented. Each approach has its advantages and disadvantages. In this section, we discuss digitized data, grid oriented representations, quadtrees, and point, line and region descriptions of terrain.

4.1.1 Digitized Terrain Data Structures

Digital terrain data (such as that available from the Defense Mapping Agency or the Engineering Topographic Labs) provides a pixel level description of features in the terrain. These maps have varying resolutions, (from 12.5 meters/pixel to 100 meters/pixel or more) and are typically obtained by manually digitizing data that was obtained via ground or aerial surveying of the regions in question. Some automated processing of aerial or satellite photography has also been employed. Terrain data is stored in terms of several (usually massive) arrays, one encoding properties of the terrain, and one for the elevation data. Digital maps provide a significant amount of

information concerning each pixel; 64 bits in the case of CATTS files of the Fulda gap region. Figure 4-1 gives a breakdown of the fields involved.

As can be seen from the figure, only a portion of the information is elevation data, while the majority is concerned with characterizing the terrain in terms of its usage type (urban, agricultural, forested, etc.), the presence/absence of significant man made features (large and small roads, bridges, etc.), and its effect on the movement of armored and unarmored vehicles (by velocity and type).

As a departure point for a more abstract terrain representation, digitized data of this kind provides several advantages; it captures a great deal of significant information, and it abstracts away a set of properties that are obviously important for a given application (military planning in the case of CATTS). The data is easily presented given a color bit-map display.

For application to tactical planning, this format has a number of disadvantages which must be overcome; it provides more data than needed (generally presenting both storage and computation problems), and it is inconvenient for representing features composed of large homogeneous regions, as well as features known to be linear (such as roads). Digital data also tends to lose a number of cultural features of import because of its resolution constraints, for example; buildings, parking areas and minor roads.

4.1.2 Grid Representations

A grid is a regular tessellation of a region of terrain (usually based on square or hexagonal tiling) in which each tile has a type and an associated set of properties. As such, a grid is a straightforward extension of a pixel view that simply employs a more convenient tiling size. (Grids are typically used as boards in wargames, both computerized and manual [Quattromani 82].) In fact, the tile properties are typically derived by averaging over the properties of the individual pixels. For example, terrain is often shown as "forested", "rough", and "clear", and average movement speeds are associated with each of these classes.

- PIXEL-LEVEL TERRAIN DESCRIPTION (25 METERS/PIXEL)
- WIDE RANGE OF INFORMATION

ELEVATION 1-16	TERRAIN TYPE 17-21	HEIGHTS 22-23	CANOPY CLOSURE 24-26	SOIL TYPE 27-31	HYDROGRAPHY 32-45	OBSTACLES 46-48	ROADS 49-51	RAILROADS 52-53	BRIDGES 54-55	MISCELLANEOUS 56-58	CROSS COUNTRY MOVEMENT 59-64

Figure 4-1: CATTS Data Fields

Grid representation often include an overlay of linear features, namely roads, railroads and rivers. Tile access restrictions representing obstacles are also often attached.

The principal advantage of a grid oriented view is that it reduces the quantity of detail involved in the underlying representation, thereby simplifying the computations over it that need to be performed. The specific addition of linear features is also more convenient that in either the quadtree or DMA formats.

The disadvantages of grid representations are as follows. First, while they are more abstract than pixels, they are not hierarchical, meaning that they provide no aid for manipulating features of interest that are larger than tiles. Second, if used in isolation, they suppress pixel level details which may be important; for example, the specific locations of clearings within tiles that are predominantly forested.

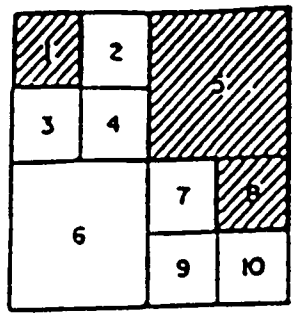
In summary, grids share the advantage of pixel representations in that they are convenient, and in part solve the digital map problem concerned with overload of detail. Their disadvantages for planning are also identical, and revolve around their non-abstract nature. I.e., both present what can be called a syntactic picture, as opposed to a "semantic" view which identifies the significant features of the terrain that truly constrain route planning and vision processing decisions.

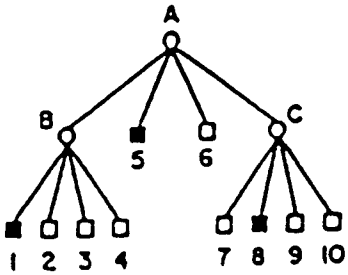
We are not contemplating the use of a grid format to the exclusion of more abstract representational forms.

4.1.3 Quadtrees

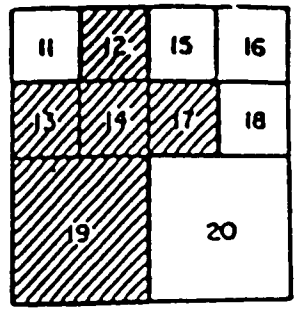
Quadtrees [Samet 83, Samet 84] are a hierarchical method for representing features in a two dimensional array of data. They operate by successively decomposing a known region into equal fourths, and characterizing each subpart according to the presence or absence of a single feature.

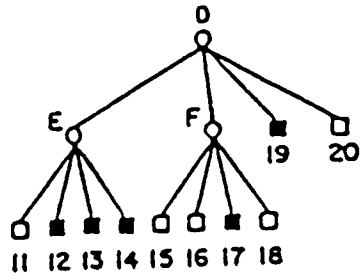
Figure 4-2 shows an example of a quadtree. In this breakdown, each subpart is



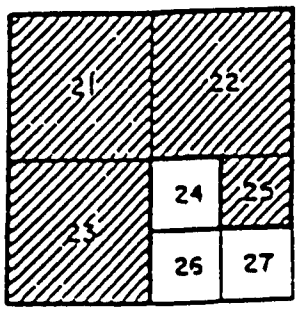


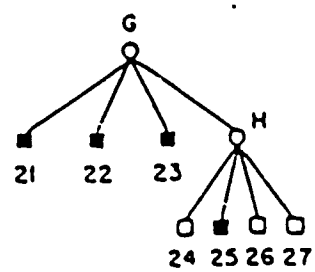
REGION I AND ITS QUADTREE



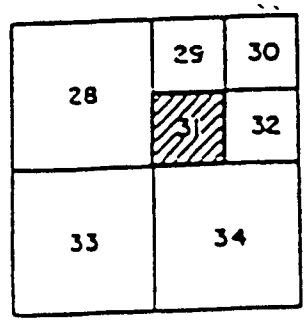


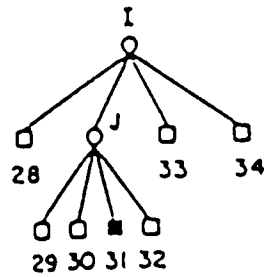
REGION II AND ITS QUADTREE





UNION OF REGIONS I AND II





INTERSECTION OF REGIONS I AND II

Figure 4-2: Example Quadtree

labeled in one of three ways; a black node stands for a region that contains nothing but the desired feature, a white node does not contain the feature at all, and a grey node represents a region that has a mixed interpretation. By successively resolving grey nodes into smaller parts, the hierarchical decomposition shown in the figure is obtained. (If necessary, this decomposition is continued to the level of individual pixels, which must be either "white" or "black.")

The principal advantage of the quadtree approach is that it supports very fast set manipulation operators (for taking the intersection and union of terrain areas). This property is derived from the fact that a quadtree is a hierarchical representation organized around the containment relation. The result, for example, is that the intersection of two pixel level features does not need to be computed if it is known that their enclosing regions are identical, or wholly independent.

Figure 4-3 show how the union and intersection operators are computed. In essence, the operations are trivial if either node is either white, or black (e.g., the union of a mixed region with a region that does not contain a feature is just the mixed region). The only non-trivial case is where both nodes are grey, in which case the problem is reduced to applying the same procedure at an increased level of resolution (there is an additional complication associated with aggregating the results of that computation).

Quadtrees also have several disadvantages. In particular,

- they must represent features in the same region of space (requiring that that region be known in advance)
- one quadtree is required for each feature (which leads to storage inefficiencies)
- they require a significant amount of computing time to define (since each pixel must be classified)
- they are inefficient (in space) for certain kinds of features (notably highly textured ones),

- UNION

	W_1	B_1	G_1
W_2	W	B_1	G_1
B_2	B_2	B	B_2
G_2	G_2	B_1	LOOK INSIDE

- INTERSECTION

	W_1	B_1	G_1
W_2	W	W	W
B_2	W	B	G_1
G_2	W	G_2	LOOK INSIDE

W = White

B = Black

G = Gray

Figure 4-3: Set Operations on Quadtrees

- they are not suited for modeling features with continuous. vs. binary values (e.g., elevation data)
- they have no localized concept of an object (i.e., objects are defined by partitioning the total space of terrain), and
- they are inconvenient for describing relations between objects.

These last two points are influencing against the use of quadtrees as a fundamental terrain oriented data structure.

4.1.4 Point, Line and Region Representations

The Engineering Topographic Laboratory has recently been engaged in the definition of what can be called a point, line and region representation of terrain which constitutes the beginning of an object oriented approach to managing terrain information. This format is envisioned as a complement to standard digitized data, which provides annotations to that data. So for example, point objects identify features (primarily cultural ones) which are beneath the resolution of digitized maps, but which are nevertheless important to describe. Examples are buildings, mountain peaks, and other significant landmarks. Linear features are roads, railroads and rivers as mentioned above. Region features (represented as polygons) are meant to capture homogeneous areas (such as pine forests, or lakes, etc.) whose pixels are identical in at least some subset of their properties. This definition allows several regions to be defined over a single geographical area (e.g., identical in ground cover type, in mobility value, soil type, etc.).

The point, line and region representation has the advantage of both capturing detail that can be lost in a grid or pixel format, and of abstracting away from pixel details where appropriate. It has the disadvantage of emphasizing polygon manipulation operations (for performing set operations on regions) which tend to be slow unless additional representational structure is supplied. (It should be mentioned that there are a variety of approaches for solving these computational problems, including the approximation of concave polygons by convex ones [Kuan 84], the use of

"polygon comparison" techniques [Weiler 80], and the maintenance of a containment hierarchy [Havens 83].)

4.2 Representation and Manipulation of Shape

Spatial models provide a mechanism for representing and reasoning about the physical shape of objects, whether they are natural, man-made, or conceptual in origin. The survey we presented in section 4.1 indicated that several types of data structures have been used for these purposes, primarily grid organizations and quad trees, but concluded that neither of these were sufficient according to our criteria. Grid oriented approaches lacked all sense of the abstraction required to construct larger, irregularly shaped features, while quad-tree mechanisms provided fast manipulation operations but lacked convenient presentations of objects.

Given this background, we decided to explore a polygon based approach patterned after the point, line and region format presented in section 4.1.4. In this view, the borders of objects are represented as polygons (really vertex lists), which might be concave or convex, and possess holes. The operations on physical shape are then cast as polygon manipulation procedures. For example, polygon intersection is used to combine descriptions (e.g., to find areas which are both hilly and forested), and polygon fill (a graphics operator) is used to display objects once they have been identified.

As mentioned in section 4.1, this polygon oriented representation has the advantage of providing a concise (and explicit) definition of objects, but the disadvantage of relying on procedures that have only been implemented in a computationally inefficient form. We decided to work with the advantages, and specifically address the disadvantages, as explained below.

The following subsections describe the details of spatial models: the representation of shape, the relationships which exist between shapes, the structure generated when shapes are instantiated in particular terrain, and the operations which manipulate them.

4.2.1 Shape Descriptors

The following types of shape descriptors are supported in this design;

- points
- lines
- regions (polygons)
- core descriptors
- boundary descriptors

The first three are basically self explanatory. Their usefulness to represent types of terrain objects is discussed in section 4.1. Core and boundary descriptors are applications of the other primitives. A core descriptor is intended to represent the central part of a feature and may be formed out of a point, line, or polygon. A boundary descriptor is a polygon, and provides the most liberal definition of a feature's extent. These shape elements are used to support gradations in the system's response; for example, the question, "does this route go through the mountains" can be answered, "yes definitely" if the route intersects the core description, and "only somewhat" if the boundary but not the core feature overlaps the route.

We are also experimenting with the use of "point set" shape descriptors, which are intended solely to fulfill a user interface role. They allow a user to define a feature by scribbling on the screen with a mouse. This results in a collection of points which can then be converted automatically into polygonal or linear regions. (There is some possibility that point sets may offer computational efficiency for computing intersection operations in some situations. If true, they will be integrated in as an internal shape type.)

Examples of shape descriptors are shown in figure 4-4. Note that the polygonal features are approximate definitions as opposed to explicit listings of all boundary pixels. This property is both desirable from a conceptual view, and important from a practical view. Since different quantities of detail are needed for specific applications,

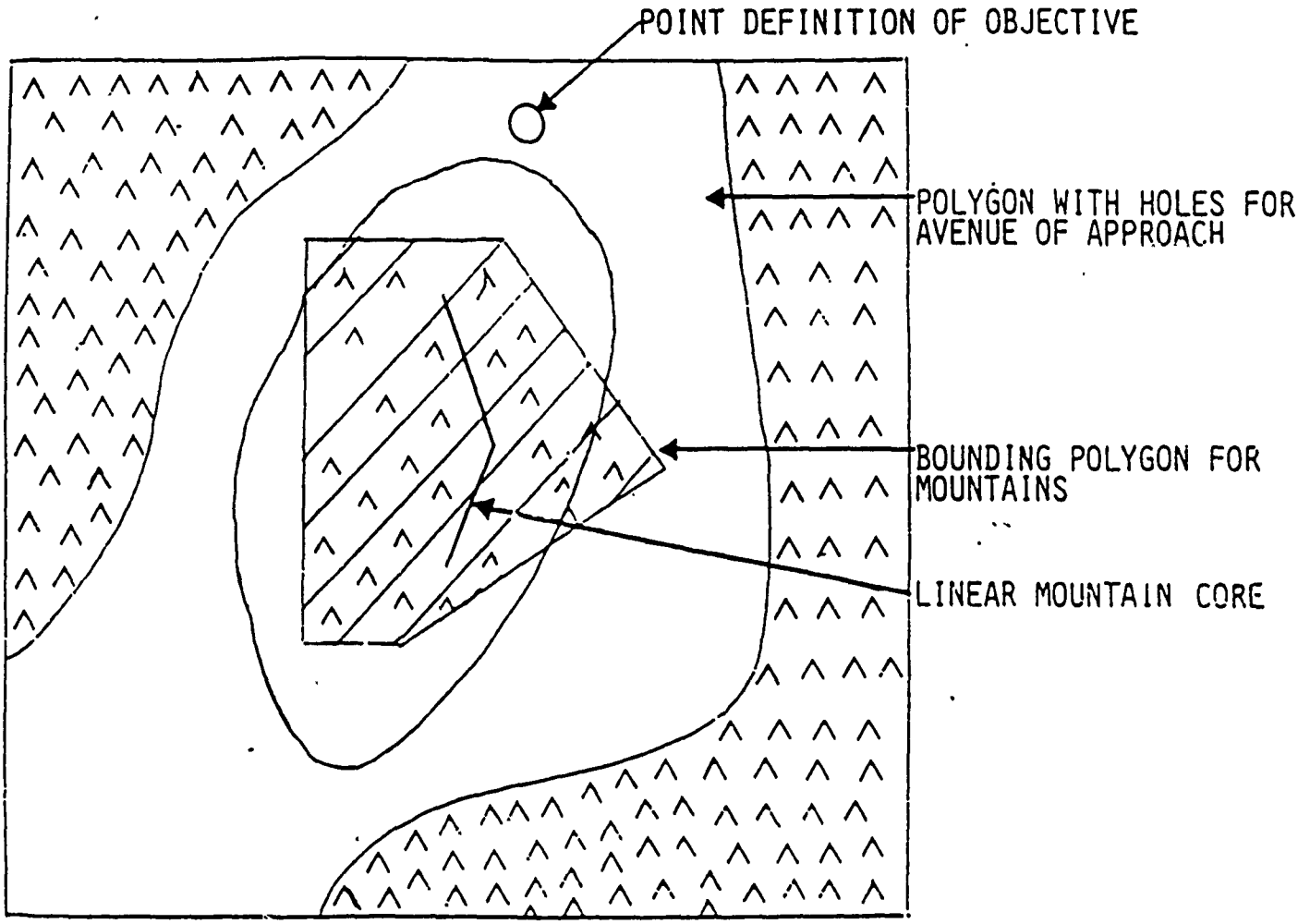


Figure 4-4: Example Shape Descriptors

we are leaving open the possibility of employing hierarchical refinements of object shapes (from very approximate to completely detailed).

4.2.2 Spatial Relations

Spatial relations identify the physical interactions between the objects in the terrain. They appear as links (in the semantic net sense) between the shape representations discussed above, and are used for several purposes. First, they support object recognition because they make the physical connectivity of terrain features explicit (i.e., they provide a framework to match upon). Second, they allow the set operations on shapes to be made more efficient (addressing the main problem associated with polygon oriented representations).

The following is a list of the spatial relations we expect to maintain. Note that all are binary predicates on shapes, and that the relations are all mutually exclusive. Those which are not symmetric posses an inverse (hence they are listed in pairs). The relations are also complete in the sense that the relation obtained by applying any set operation to any two polygons is a member of the list below.

- disjoint
- overlap
- contiguous
- in, contains
- encloses, enclosed-by

Figure 4-5 shows these predicates as defined over polygons, although they also operate, with some exceptions, on point and linear shapes. The exceptions are overlap (which is not defined between points or points and linear shapes), in and contains (which do not operate between points), and the encloses, enclosed-by pair (in which the enclosing object must be a polygon). The concept of enclosure here is very similar to that of containment; enclosure is used in situations where objects have holes. Enclosure does not imply containment.

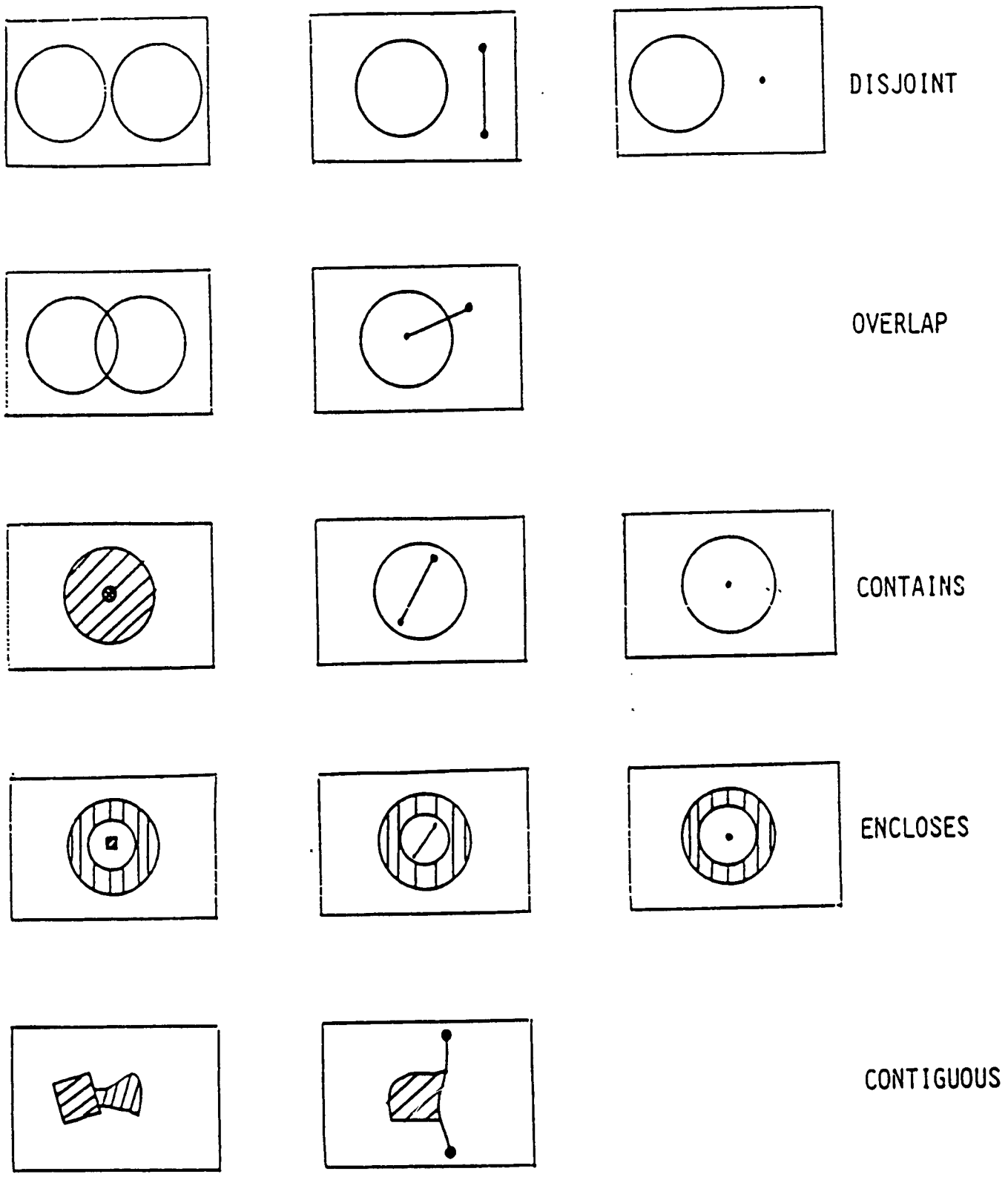


Figure 4-5: Spatial Relations

It would be possible to take an alternative approach and define links for the union, difference and intersection relations (which generate the other set operations), and use them to connect all polygons instantiated over the terrain. This format could indeed be used to encode the relations described above, but at the expense of clarity in highlighting the named operations and by causing an increase in the complexity of the network interconnecting shapes. For example, disjointness would have to be expressed by stating that the intersection of A and B is null (also true of contiguous objects), and that the union operation results in two distinct objects. Similarly, containment becomes the fact that the intersection and union operators chose one polygon or the other, and that the difference operator creates an entity with a hole in the middle. This complexity tends to obscure the property of simplifying further polygon manipulations.

Several possible additional relations are "near", "through", "before" and "after". Nearness implies a fuzzy distance metric which is dependent upon the scale of the features, while the "through" relation captures the common situation in which a road goes through a polygonal region. (Throughness between polygons would identify regions that "cut across" one another.) The relations "before", and "after" address the fact that routes, lines of advance, and avenues of approach all have a direction. These relations would be helpful in answering questions of the kind, "what are the first obstacles encountered along the planned path?".

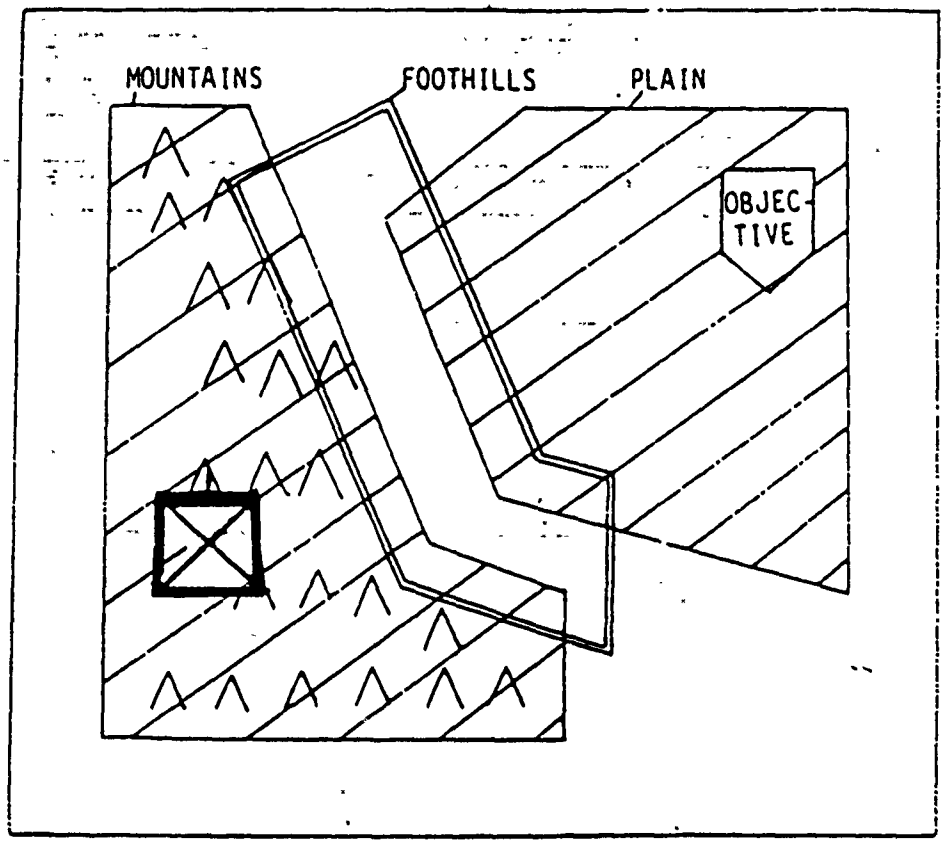
It should be mentioned that both the containment and enclosure links can be used to form a hierarchy of objects in the environment. This has an important consequence for the application systems which rely on the terrain reasoning kernel; in specific, the hierarchy provides a quality of abstraction in reasoning where the properties of larger objects can be examined without regard to the details of the terrain features which they enclose.

4.2.3 Networks of Spatial Relations

The process of instantiating the relations discussed above creates a network of links connecting the spatial components of objects in the terrain. It turns out that the properties of the spatial relations allow some economy in that structure since several obey transitive, and distributive laws. This allows some relations to be inferred, vs. computed and explicitly stored (see [Shapiro 85a] for details).

Figure 4-6 shows a network of relations that might exist in a simple scene. This network is relatively sparse (meaning that all polygons have not been compared with all others) but it illustrates the fact that a number of additional relations can be inferred. For example, given that the mountain is disjoint with the plain, an automated system can answer "no" to the question "is the objective IN the mountains?" by inheriting the disjointness relation. The same process can be used to realize that the infantry battalion is also disjoint from the objective, since disjointness inherits across any number of containments. On a similar note, two objects have no intersection if they are enclosed or contained in any shapes which are contiguou Another principle is that no object can overlap another unless they have the same parent (containing or enclosing shape), or unless two of their parents overlap. From a computational perspective, these types of considerations save a great deal of effort.

In terms of knowledge acquisition, this structure of relations can be built incrementally as new shapes are defined. We have not identified the appropriate procedure yet, but three possible strategies come to mind. The first is to avoid all exploratory comparisons aimed at building a classification net and only compute those which are required. As time progresses, this should result in a network that simplifies further operations. The second strategy is to determine the most useful relations on the theory that a small initial investment will save a larger number of comparisons later; computing the smallest existing polygon a new shape is IN or ENCLOSED-BY would at least identify its place in the hierarchy and allow superseding disjointness to be inherited. The third strategy is to develop a minimal classification which provides the greatest simplification when further operations are required. This would involve



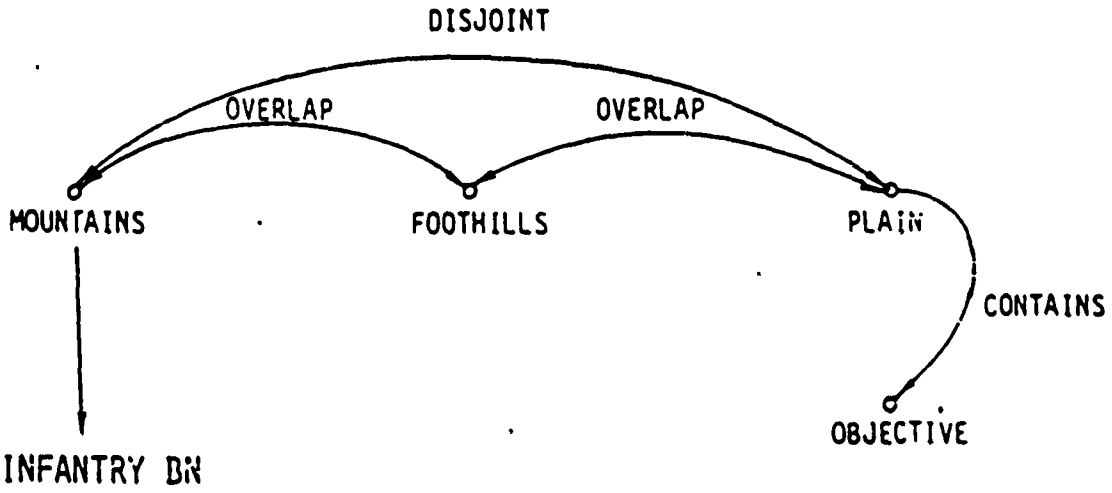


Figure 4-6: An Example Network of Spatial Relations

principles like building an encloses/containment hierarchy, pushing disjointness connectives to the highest possible place in the tree (since they inherit down), and conversely placing overlap links as low as possible (since the containing objects must also overlap until a common parent is found).

4.2.4 Set Operations on Semantic Objects

We have discussed several methods in this chapter for computing the results of set operations on the spatial component of objects, but there is an open problem in deciding what it means to intersect (etc.) the associated symbolic descriptions. That is, when objects are combined, what set of properties should be attached to the end result? Some of our thoughts on this topic are discussed below.

If we consider the purpose of the operations on object shape, there are primitives for aggregating shapes (union), identifying shared regions (intersection), and noting discrepancies (difference). If we view the semantic component of objects as attachments to shapes (i.e., as descriptions of physical regions) then the role of object manipulation is to compute a semantic analog whenever a spatial operation is applied. In this view, the goal of intersecting objects is to determine the properties that still apply to the intersected region. Similarly, the goal in union is to build a composite spatial region and determine which properties to ascribe to the larger whole (i.e., an abstraction process). The goal in computing difference is similar to the intent of intersection; it is to identify the properties that apply to one object and not the other.

It is important to notice that this approach creates two types of objects which define geographical and non-geographical concepts. Geographical objects describe regions of terrain (a hill, a valley, an avenue of approach) and are operated on as above, while non-geographical objects have no intrinsic terrain analog. A mortar, an infantry battalion, and a SAM installation are examples of the latter. Each has properties and parts (the radar and missile components of a SAM, its range and accuracy parameters) but these properties cannot be combined in the same way. That is, the range of a SAM and the number of vehicles in a Tank Column cannot be intersected, although their associated terrain regions (a SAM site, the location of the tank column) can.

Figure 4-7 shows an example of object combination applied to an infantry site and a mortar field of fire. When the objects are intersected, the result is an object which is a-part-of both terrain features, meaning that it will inherit properties from each. The question, "what is the concealment within the fire-covered infantry site?" can then be answered as "good", while its density of fire coverage can also be derived (via inheritance) as "high". Object difference results in a region which is only a-part-of the infantry site; meaning that the question "what is the density of fire?" is given a null answer. It is also important to notice that this mechanism for combining objects relies on the condition that object properties uniformly describe their terrain regions. A property attached to the infantry site listing the "number of contained companies" would be inadmissible as it would not behave appropriately under set intersection and difference. Those kinds of measures must be associated with the infantry object itself, or recalculated whenever object combinations are required.

Object union is better illustrated in the situation depicted in figure 4-8. Here, three route components (all of which have similar property lists) are aggregated into a single entity. The resultant object has the combined spatial region, its subparts are obvious, but there is a question as to which properties should be included. Union objects implies intersection of properties, meaning that the risk value of the route should be listed as "safe". However, the question "what is the transit speed of the route?" deserves some answer other than enumerating the travel rates associated with its parts. The right answer is to characterize the route as a whole as *go*, *no go* or *slow go*.

The issue is then how to control abstraction on properties. We have no explicit answer at this time, although we are considering including explicit procedures in object models for directing "upward inheritance" of specific properties; that is, to determine how speed, safety or other characteristics are aggregated from component parts to a whole.

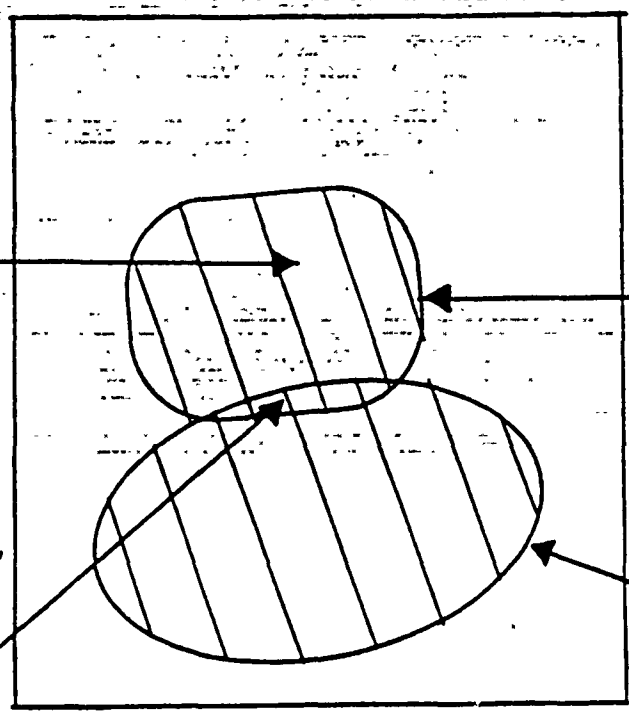
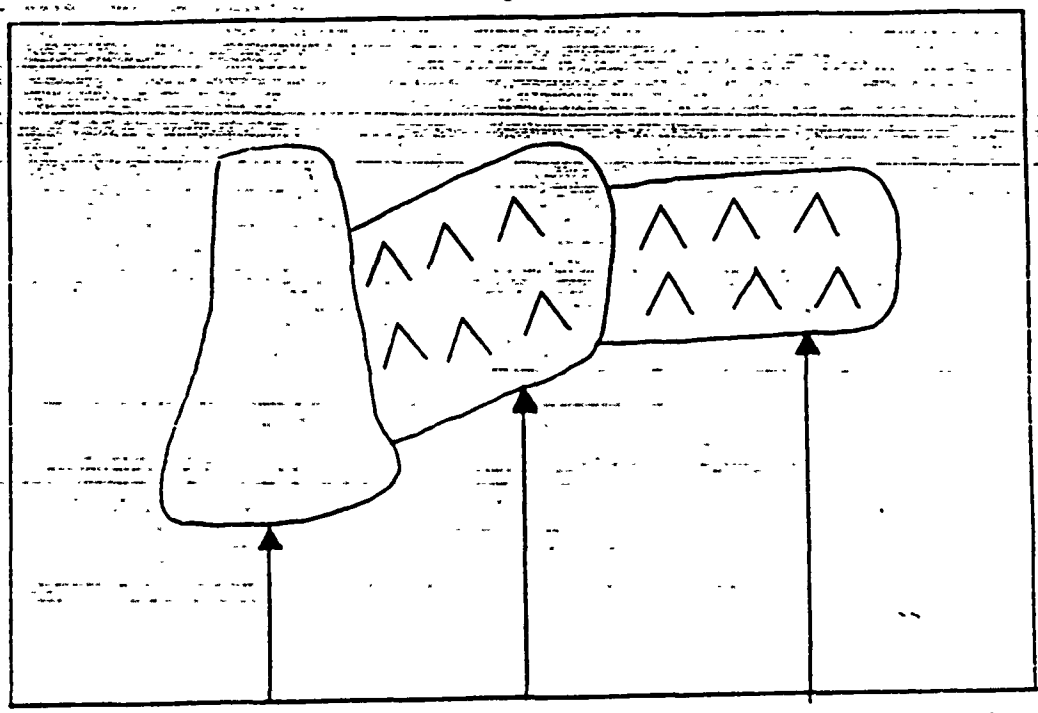


Figure 4-7: Object Intersection and Difference



PLAINS ROUTE SEGMENT
- SAFE
- FAST

MOUNTAIN ROUTE SEGMENT
- SAFE
- SLOW

FOOTHILLS ROUTE SEGMENT
- SAFE
- FAST

COMBINED ROUTE
- PARTS (PLAINS ROUTE, MOUNTAIN ROUTE, FOOTHILLS ROUTE)
- SAFE
- SPEED?

Figure 4-8: Object Union

4.3 The Symbolic Map Presentation System (SMAP)

Partly in order to gain experience with terrain data, we developed an early prototype system (SMAP) for presenting and manipulating CATTS data (Shapiro 85a). This system had several interesting features worth noting here; it provided several operations on elevation data, a form of a multiple overlay color display, and the beginnings of a feature extraction capability for identifying regions of military interest. (No automatic feature recognition was included).

The architecture of SMAP was quite simple (see figure 4-9). It operated on CATTS data, which is a 25 meter resolution grid giving 64 bits of feature data and 16 bits of elevation data for each point, which in our case covered a 10 km by 14 km area of the Fulda Gap in Germany. For operations, the user could request fields from the CATTS database by mousing on menus, display them on the color screen, and invoke certain algorithms on the elevation data described below.

With respect to the color display, SMAP employed a predefined sorting of CATTS fields into background and overlay; space-filling CATTS fields such as soil type or ground cover were background, and non space-filling fields such as road nets, river or obstacles were overlays. Overlay data could be placed on top of any background display (allowing the user to view road nets over soil type, for example. as a precursor for building traversability displays). This turned out to be quite valuable as a visual presentation aid.

The user could invoke line of sight operations by clicking on any point of the display, making it possible to correlate observability or fields of fire with selected terrain conditions. Also included was an ability to evaluate the risk of a proposed path by essentially forming a histogram of the route's observability from different vantages. A final operation was a manual feature extraction capability; here, the user would define a particular shape as a named feature (for example, a hilltop objective) which could then be displayed in the context of selected backgrounds and other overlays.

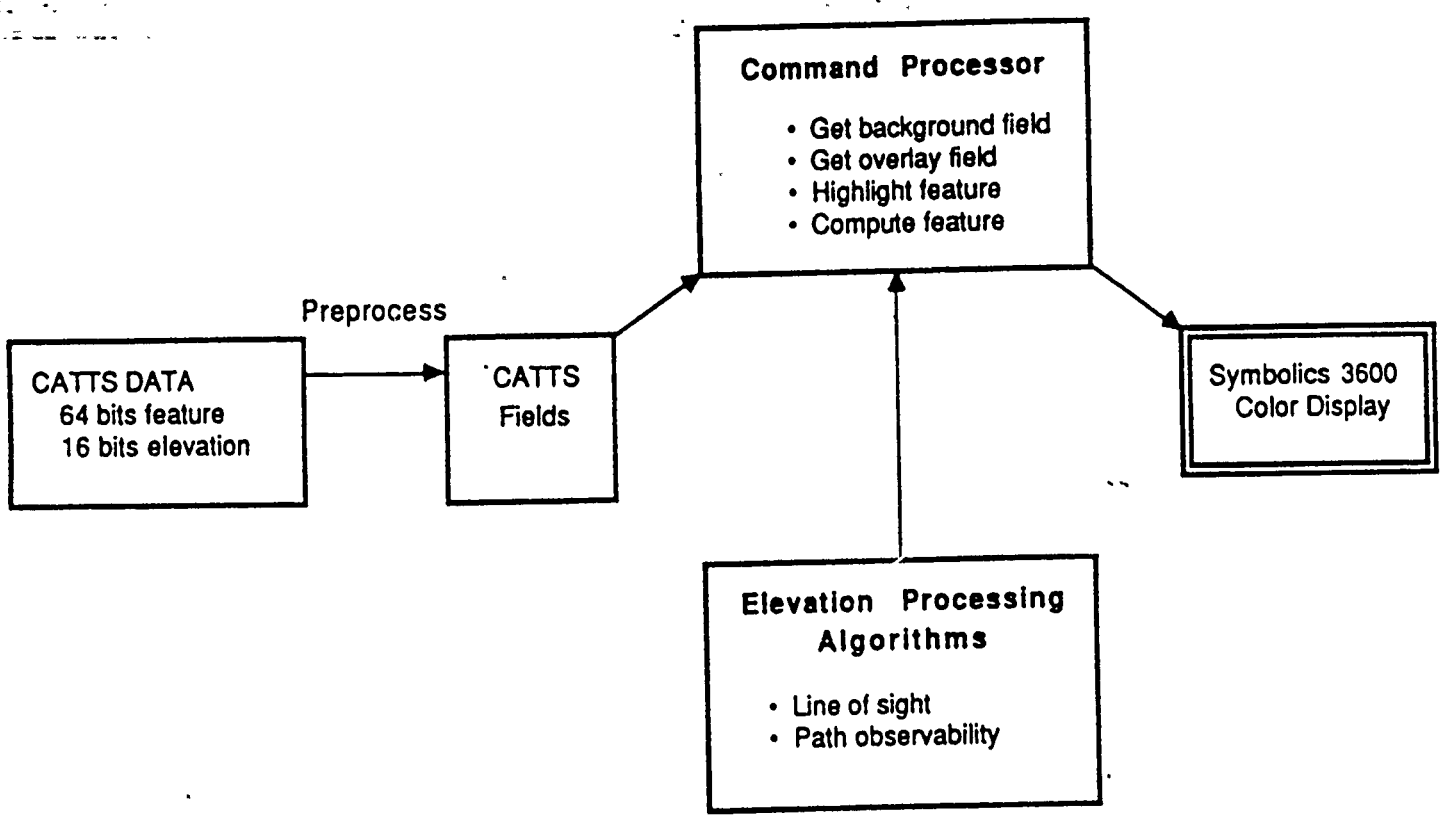


Figure 4-9: The architecture for SMAP

On the whole, the facilities of SMAP were quite limited, but its creation taught us several important lessons; that multiple overlay presentation was the preferred form of terrain display, that the supporting implementation techniques were non-trivial, that there are a class of operations on elevation data which are desired features of any terrain analysis system (calculating direct and indirect field of fire from regions or points, and the inverse operations in terms of observability), and that the task of *automatically* identifying features of military interest was the critical thing to pursue. Our development of TAPS was specifically directed towards that ability.

We ultimately abandoned the implementation of SMAP for very technical reasons; it was deeply tied to CATTS data formats, it's method for supporting multiple color overlays required preprocessing of all terrain data, and it had very little flexibility in terms of combining overlays, including a complete inability to select display colors.

35 millimeter slides shown output from SMAP are provided in appendix II.

5. The Design of TAPS

This chapter describes the capabilities, design, and initial implementation of the Terrain Analysis and Planning System (TAPS). This system demonstrates methods for supporting battalion (and other echelon) commanders as they construct plans for combat situations, and is motivated by our study of military operations which concluded that terrain reasoning was a critical area to pursue. Specifically, TAPS interacts with a user to identify features of military interest from digitized map data, selects features appropriate for particular tactical actions, and evaluates the impact of terrain on actions proposed by the user. This defines a system with the potential for helping the S2 officer satisfy the terrain analysis requests typically generated by the S3. As such, it is not a true planning aid (which would include a representation for combat plans, an appreciation of time, knowledge about the resource allocation process, and an ability to suggest tactical alternatives, etc.), but rather a system which supplies the critical information such an aid would require.

5.1 The Motivation for TAPS

In order to better define the capabilities of TAPS, it is important to examine the way terrain analysis is employed by the military in more detail. From the introduction above it is clear that its role is to support planning, but we can motivate our approach further by examining the tasks terrain analysis specialists are called on to perform.

From our observations, requests for terrain workups come in two forms; the first is to produce a more or less generic analysis of the entire battle area which identifies obstacles, cross country movement rates and weather effects on movement if special weather conditions are expected to occur. (This list is not complete, and seems to differ somewhat by echelon). The result is a collection of acetate overlays which can be created as soon as the battle area is identified, often well ahead of any specific engagement (years in the case of anticipated conflict areas such as Fulda). The second type of terrain analysis request is motivated by the desire to support specific tactical actions and occurs more frequently during the operations cycle. The essential feature of

these requests is that they require a large amount of context information concerning the units, equipment and tactical actions proposed. Examples are to select sites for river crossings, areas for engaging a moving enemy, helicopter landing areas, mine field emplacements, locations suitable for prepared defenses, and avenues of approach for friendly movements, etc. To give a flavor for the amount of context information required, consider a river crossing operation; any hypothetical terrain analysis system will have to know at least the following:

- the kind of equipment crossing the river (to determine whether fording or bridging is appropriate, and the locations appropriate to each)
- the kinds of mobile bridges available
- the permanent bridges already in place (a primitive terrain feature)
- the distance of hostile units (if the scenario is move to contact, fewer protective measures are required)
- the defensive weaponry available to the operation (given artillery, observation over the opposite bank is less essential)

Given this input, the output of the river crossing terrain analysis is then a list of features similar to the following:

- an identification of where to cross the river (and how - via bridges or wading)
- identification of ingress paths to the river, and egress from the opposite bank
- identification of regions that provide direct fire cover to the crossing point
- identification of points that provide observation over the opposite bank
- identification of areas providing cover for the expected enemy (i.e., predications of likely enemy positions)
- a description of river bottom conditions and soil conditions leading to a discussion of throughput of forces (conducted more probably by army engineers)

Note that the output above is expressed in terms of terrain regions that fulfill

specific tactical purposes. A critical point is that these can ultimately be generated by applying the right set of low level queries to the underlying data concerning the terrain.

In summary then, the function of terrain analysis is to take an expression of a planned operation and translate it into a list of the terrain features which can fulfill the different roles in the plan (in the case of a generic cross country movement analysis the notion of a tactical operation has to be taken somewhat loosely). This is exactly the capability we are addressing through our research in the BCA project. The TAPS system (described at length in this chapter) provides basic support for this terrain analysis activity in terms of data sources, query, and display tools. It should be clear however, that the eventual solution will also require models for tactical actions and terrain features of interest, in addition to a knowledge of the equipment capabilities and organizational units in the military domain. (Our thoughts on the structure of these models is discussed in section 5.5.2.)

While we have always targeted terrain analysis as the TAPS application, we have also been concerned with the underlying question about the extent to which terrain analysis subsumes combat planning. As such, we have a strong motivation to examine realistic problems. Towards this end we examined scenarios from the military training literature, identified a mission and a notional battalion (with realistic equipment list) to set context, and extracted some number of terrain analysis queries which were motivated by that scenario. The results (see [Shapiro 86]) tend to confirm our belief that the use of TAPS (or its successors) will strongly motivate battalion level combat planning decisions. In addition, we feel that the TAPS capability is important to the echelon of corps, division and brigade.

In the remainder of this chapter we describe the TAPS architecture, the Multiple Pane Interface (or MPI), the GINDB subsystem, and the World Model. Appendix III provides a set of color slides and a description of output obtained from TAPS. We have omitted reproduction here of the battalion scenario and equipment descriptions presented originally in [Shapiro 86].

5.2 TAPS Architecture

This section provides an overview of the architecture of TAPS, which is an interactive tool for supporting terrain analysis. The task in terrain analysis is to identify regions of space that satisfy the requirements of particular tactical actions. When performed by the military, the input data of this process is typically aerial photography, paper maps (or their electronic analogues), and possibly a geological analysis or a digitized map providing elevation and simple feature data (although the later is available only for restricted areas of the world). The output is a collection of acetate overlays which identify and interpret important terrain regions when placed over an underlying map.

TAPS supports the same type of I/O behavior described above. For our purposes, it is reasonable to rely on digitized map data as input (elevation and feature data both), and we have defined an output mechanism which supports multiple overlays analogous to (and in many ways more capable than) the acetate procedure currently employed. This in turn provides a very convenient and powerful user interface format. With respect to the process of terrain analysis, our basic design decision has been to view it as an interactive feature extraction procedure in which the user grows a database of interesting regions by applying a series of questions to the underlying information representing the terrain. To support this approach, we have introduced the concept of feature models into TAPS, which define the component parts and constraints on terrain regions that fulfill tactical roles. We hope to use the act of instantiating these models as the basis for a mixed initiative interaction between the user and TAPS.

This approach gives rise to the architecture presented in figure 5-1. There are three major components of this system; a language for constructing terrain based queries (called GINDB for Geographically Intelligent Data Base), a display interface (called the MPI for Multiple Pane Interface) for presenting candidate regions to the user, and a knowledge base (labeled the World Model) that contains the tactical feature models mentioned above in addition to data concerning the organization and equipment of the military domain.

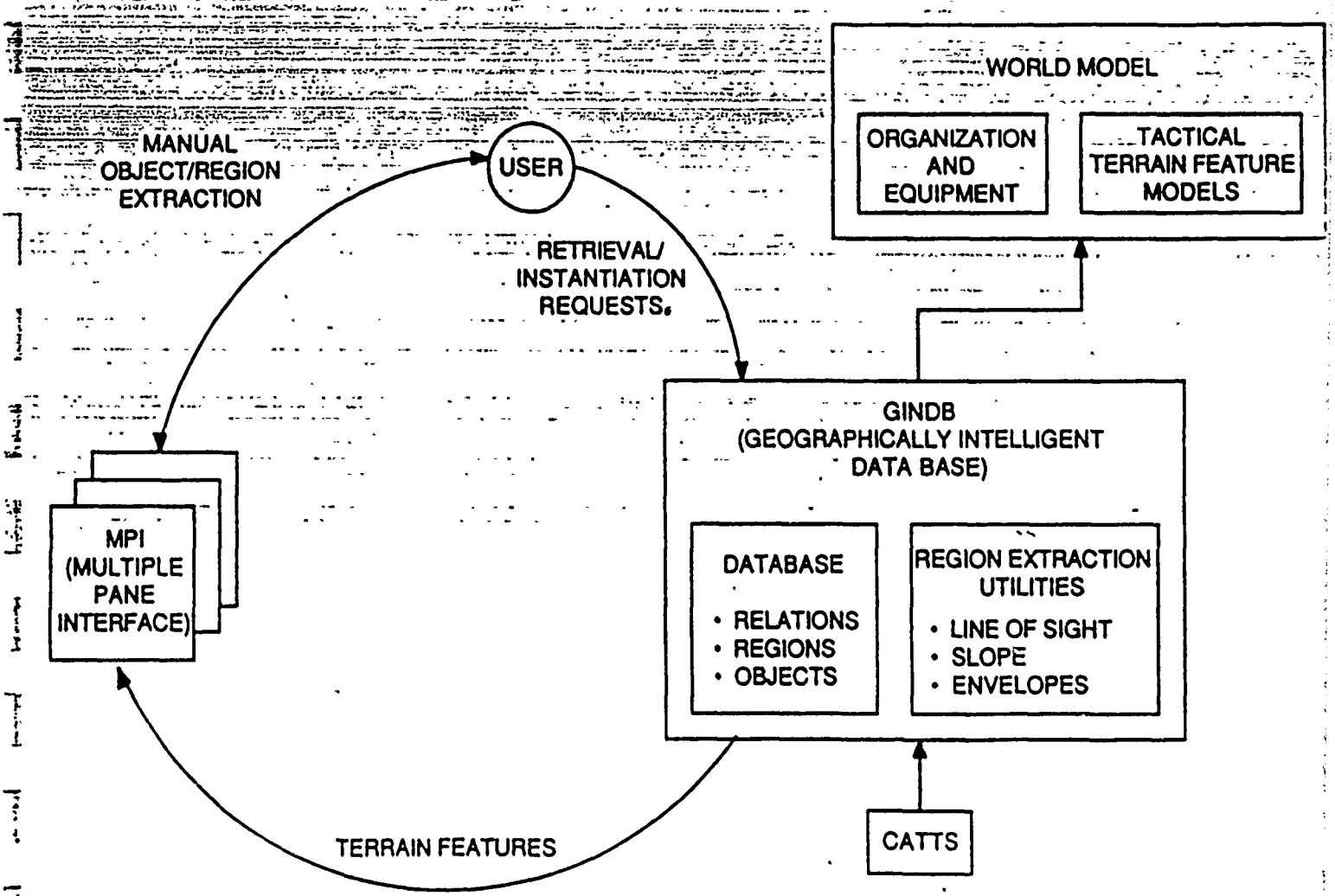


Figure 5-1: TAPS Architecture

An example of information flow through the diagram is as follows. Assume that the user decides to develop an avenue of approach for a heavily mechanized battalion task force. He might start by building a GINDB query to retrieve the region that is basically high mobility, i.e., the AND of flat, not forested and not urban. (GINDB answers this by applying the appropriate logical operators to fields extracted from the underlying CATTS data.) After displaying the results, he might extract (manually, by drawing on the screen) the portion which is between the current location and the objective, and store this as the first approximation to the avenue of approach in the database maintained by GINDB. Next, he might decide to reduce this area by eliminating the portions not navigable by tanks due to weather or soil conditions. This could be accomplished by generating an overlay for the terrain regions with non-porous, clay based soils that are also in the current avenue (a low level GINDB request) and displaying it on top of the high mobility zones retrieved above. At this point, it would be possible for the user to eyeball and then adjust this restricted avenue (again manually by drawing on the screen with the mouse, or semi-automatically by requesting further manipulations of the region on display via GINDB) in response to some criterion that he possesses external to the machine. An example is that the particular vehicles involved might have very good traction characteristics, so only the least tractable areas near bodies of water need to be removed. At this point the analysis might progress to the more detailed step of identifying river crossing sites, which would involve instantiation of the 'river crossing site' feature model outlined in the previous chapter. Our view is that this would proceed in stages (just like the avenue of approach scenario described here) except that the interaction would take place using the vocabulary of the component parts and optional/required parameters defined by the model. Ultimately, the same type of low level terrain queries would be processed by GINDB.

The following sections describe the major components of this architecture in more detail. Before continuing, we should point out that GINDB and the MPI have undergone significant implementation, while organization of the World Model lives still very much at the level of design. Some of the most interesting technical issues in the project revolve around determining the appropriate composition of these feature models,

and we will be exploring this topic throughout the remainder of the contract. Our current ideas are presented later in this report.

5.3 The Multiple Pane Interface (MPI)

The MPI (Multiple Pane Interface) is a flexible tool for displaying data which comes in the form of multiple color overlays. It is an implementation of the metaphor of holding multiple panes of glass up to the light, where each pane is colored in the areas where it contains data (see figure 5-2). The resultant display is what the eye would see as the various features overlay one another. The MPI provides operations for setting the color and overlay mode of individual panes (e.g., hashing, outlining, etc.) and for reordering the sequence in which the panes occur. In addition, there are operators for describing the contents of a pane (or subset thereof), returning the form used to generate the pane, and input procedures for extracting data from panes and for creating new panes manually (via the mouse).

At its core, the MPI is a color graphics tool which is independent of the BCA application. However, in the context of BCA it can be used to compose terrain displays for such things as cross country mobility (which includes obstacle, slope and soil criteria), order of battle data over a feature map, and avenue of approach displays which include elevation contours, shaded areas for zones of critical terrain, and outlines of objectives. The MPI mouse input capabilities support placement of icons, and addition of annotations such as labeling the axis of an enemy thrust, numbering hills, or flagging terrain alterations such as mine fields. The overlay capability can be used to display output from terrain analysis procedures which compute new features from the underlying data, for example, to show fields of fire on top of an order of battle display as an aid for evaluating force positions.

5.3.1 Approach

As a graphics problem, there are two issues involved in constructing the MPI. The first is to define a logical mechanism for displaying multiple features over a given point such that they are visually distinct and overlay appropriately. The second (and

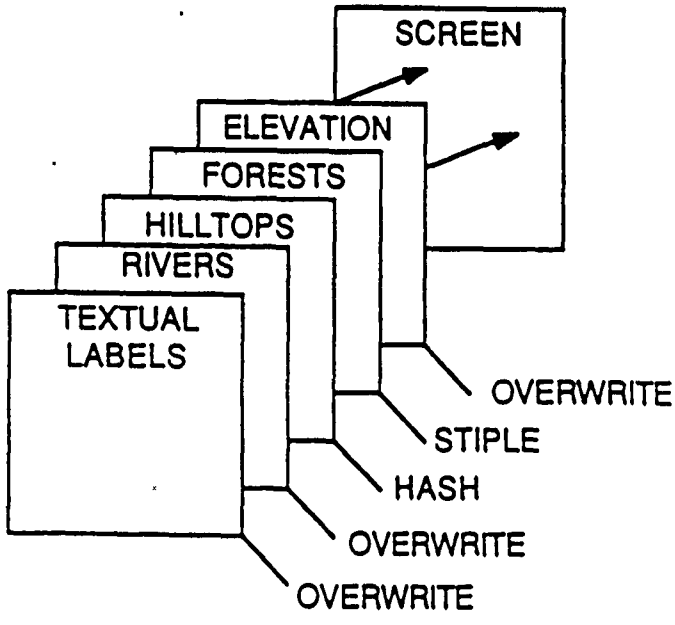


Figure 5-2: The MPI Metaphor

obviously related) problem is to implement this mechanism efficiently, since a simple analysis shows that the volume of data involved in processing multiple overlays is taxing at the level of machine cycle times. Even for the comparatively loose constraints of a research prototype, clever algorithms are required.

This speed requirement can be illustrated as follows. In our formalism, an overlay consists of a single feature which is represented as a binary image and displayed in a single color. For the purposes of calculation, we can take 30 overlays as the practical maximum, since visual clutter effects are clearly dominant at that level of complexity. Our digitized map data employs 25 meter resolution, so a 512 x 512 grid (which is 20 kilometers on a side - about twice the size of a typical battalion area of interest) contains ~262,000 pixels, and 30 planes carry close to 8 million bits. If these are accessed individually, merely touching them (reading and writing) requires ~16 seconds at 1 microsecond cycle times (using a 2 instruction loop that leaves little room for additional processing). This should make it clear that any practical implementation has to manipulate the overlay data cleverly, and operate fairly closely to machine specifics.

We have solved the above efficiency problems by employing a number of assembly-language programmer's tricks; we reference memory 32 bits at a time and employ the associated word packing and unpacking techniques, we have made extensive use of microcoded operations such as *bitblt*, and in situations where no microcode support was available we have written tight assembly language loops (via the system-internal calls available in zeta-lisp). In addition, we have introduced an intermediate data structure called the *plane cube* which represents parallel bit slices of the 8-bit wide color screen memory. The mechanics are explained in the section 5.3.5.1, but the effect is to support extended use of bitblt. While this is an undeniably arcane implementation decision, it appears to produce the desired speed; we expect to process the 30 overlay test case in circa 2 seconds.

The procedure we have adopted for combining overlays is actually quite simple (see figure 5-3). It relies on two central data abstractions; *overlays* and *aggregates*,

where overlays correspond to individual panes of glass in the MPI metaphor and aggregates are ordered sets of those panes, and therefore the source of data for the color display. In our approach, the data from each overlay is sampled in accordance with some *pattern* representing an overlay mode (e.g., hashed lines, a matrix of dots, etc.), and the resultant data mask is written to the color screen memory using the desired color. This is done for every overlay in the aggregate, producing a multi-color display. The net result encodes multiple features over a given area by displaying some pixels from each.

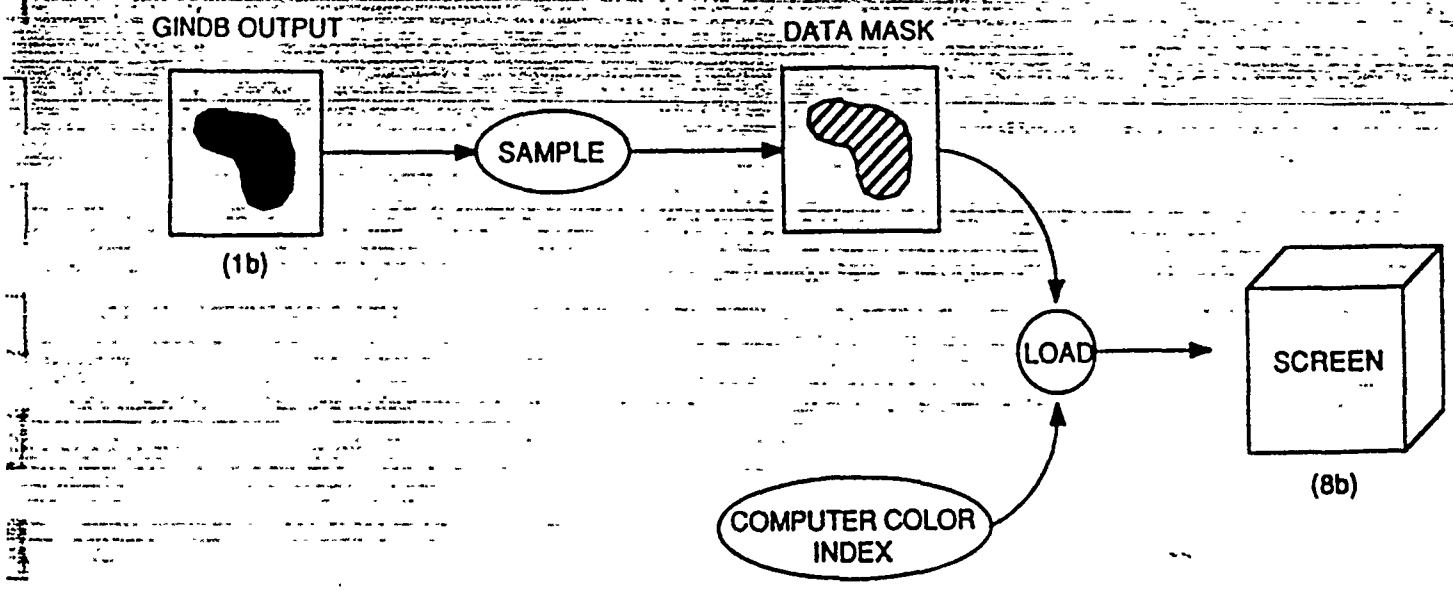
It is worth noting that this approach limits the possible overlay modes to the ones expressible as sampling operators. In order to answer the request "display the hills over the background data by tinting the background red" we would have to introduce a different type of operator which compares the color associated with the source pane to the ones already on the color screen, and then invokes a combination function such as color averaging, or color wheel addition. We have made provisions for incorporating this type of operator (called *translucency*) in our design, but since the obvious method of implementing it (by calling a function on each source/destination pixel pair) is too slow to be acceptable, and the more efficient approaches are correspondingly involved, we have not included the operator in our current implementation.

5.3.2 Overlay Modes

The planned set of overlay modes are therefore as follows;

- overwrite
- hashing
- stippling
- borders
- translucency

In *overwrite* mode, the source data is not reduced by sampling, meaning that it completely obscures whatever area is beneath it. *Hashing* is self evident, and *stippling*



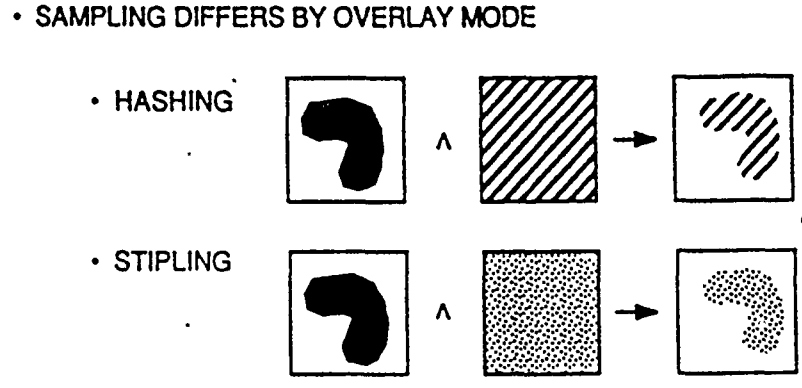


Figure 5-3: Computation of Overlay Modes

refers to a sampling operator which takes some percentage of the pixels in the source. Our current implementation supports about four sampling frequencies ranging from 5 to 50%, and our experience is suggesting that higher sampling frequencies will be more satisfying to the eye. The *borders* mode produces outlines of area features instead of sampling their contents. It turns out that this can be implemented quite efficiently by a kind of parallel edge detection operator based on the microcoded bitblt function.

The *translucency* operator identifies a class of binary predicates on colors, and therefore cannot be implemented via unary sampling. It is intended to work in conjunction with a combination function, such as color averaging or color wheel addition, to support the effect of looking through a tinted piece of cellophane. We have observed that stippling and translucency tend to be perceptually equivalent since the human eye averages colors that are interspersed so tightly. An explicit translucency operator will simply provide more control over the color of the result. Implementations for translucency are discussed in section 5.3.5.2.

It is possible to produce a wide variety of overlay modes of the kind described above. Since they are defined solely by a sampling pattern, they are trivial to construct.

5.3.3 MPI Architecture

The architecture responsible for this behavior is shown in figure 5-4. It consists of a small collection of data abstractions whose operators are directly accessible by the user. Two main abstractions are involved; *overlays*, which define single panes of glass in the MPI metaphor, and *aggregates* which collect and order overlays and are ultimately displayed. The roles of the remaining support abstractions are as follows; the *color table manager* allocates colors (really indices into the hardware supported color table) from the 256 member set available on the Symbolics Lisp Machine at any given time, and the *pattern manager* defines individual overlay modes. The *data source* is an abstraction representing the source of the binary images incorporated into overlays. Its purpose is to separate the MPI from whatever system is being used to drive it, in this

case, GINDB and the remainder of TAPS. The operations on data source objects provide the binary image itself, or a form which produces that image. The latter is the original query to GINDB.

At the current time, the operators of the overlay and aggregate abstractions are directly available to the user. For example, on receipt of a dataset from GINDB, the user can *define* an overlay (which amounts to running *set overlay mode* and *set color*) and then *display* it (which uses *return data source* to get at the input feature map). Since the design allows many aggregates to exist simultaneously, we have added the overlay operation *aggregates which use* in order to provide the necessary indexing.

Most of the operations on aggregates are self-explanatory with the exception of *extract new overlay* and *describe data in region* (both of which are partially implemented at this time). The first is the mechanism for using mouse input to create a new overlay by drawing on the screen. The intent is to maintain a blank top pane which is always available for creating such annotations, but is treated specially by the architecture in two ways. First, it will have the equivalent of a simple (single color) paint program attached to support user 'doodling'. Second, annotations to the pane will be displayed immediately instead of processed through the multiple overlay mechanism. This shortcut is possible (aside from being absolutely necessary from a user interface perspective) because the annotation pane is outermost, so by definition its data cannot be occluded by anything else (so processing for additional overlays is not required).

The describe function is available as a consequence of maintaining the data source as a separate functional object. At any time, the user can mouse on the screen, and this operation will determine the overlays who's data is displayed over that point (there may be more than one). The result is a list of overlays (with mnemonic name), and in our case the GINDB queries which produced that data in the first place. Modifications on this operation can be used to describe the features in a given region.

The data structures which underly these abstractions are also quite simple. Each

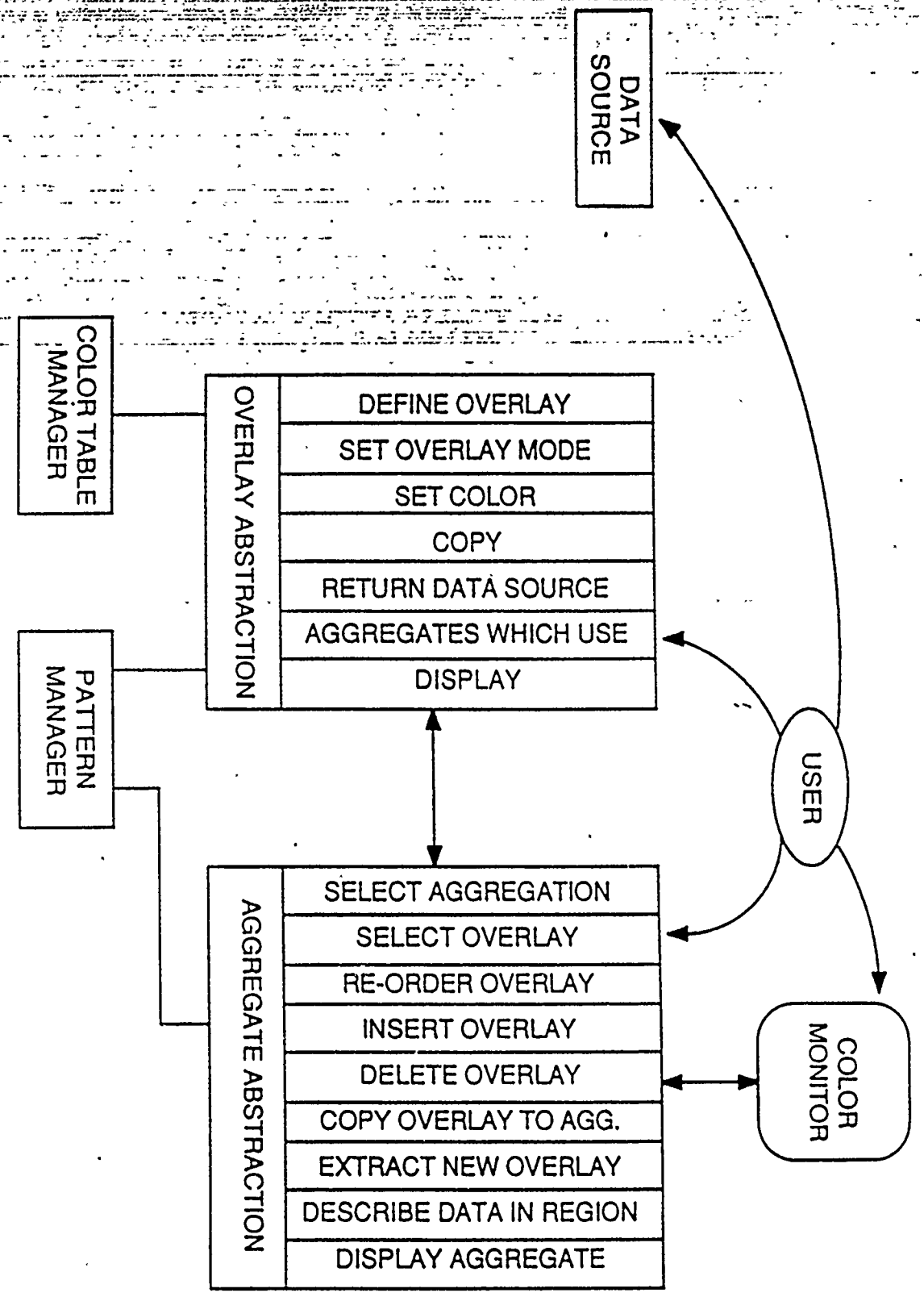


Figure 5-5: MPI Architecture

overlay contains four parts (ignoring the slots which are relevant only if one is reading the code):

- a data source object (which contains a binary feature map)
- a color
- an overlay mode (or *pattern*) used to display it, and
- a patterned-bitmap, which is the displayable form of the input data after the overlay (or sampling) pattern has been applied.

Colors are represented as numbers which we treat as indices into one of the four color tables (which associate numbers between 0 and 255 with color specifications manipulated by the Symbolics hardware). Patterns are bitmaps used to sample the feature data before display. They encode hashed lines, herring-bone patterns, various dot densities (used for stippling), and others. (See the section on overlay modes, above).

Aggregates contain two parts:

- an ordered list of overlays, and
- an image of the color screen window (initially blank) in which it is displayed.

The ordered list allows overlays to be projected on the color screen in sequence, while the screen image provides a stable backup copy should the actual display be altered (as it will in the painting mode when the user constructs new overlays). Note that since each aggregate knows the overlays it contains, and each of those overlays knows its assigned color and display source, the aggregate's *describe data* operation has access to the information which will allow it to identify the overlay name and query responsible for each pixel on the screen. We expect this to be a powerful user interface function, and plan to use it heavily in comparing regions selected by the user with regions produced by TAPS to fulfill the same goals.

The only other data structure of import at this time is the *query response structure* which underlies the *data source* abstraction that interfaces the MPI to GINDB. It has three parts:

- a binary image encoding the GINDB result
- a list of the *things* returned, if any
- the query form which produced the GINDB output

The distinction here is that GINDB can return either regions or symbolic objects (things) depending upon whether the query form was *retrieve-region* or *retrieve-thing*. In the later case, the query response structure holds the list of objects returned (so that the describe functions of the MPI may access them), while the binary image reflects the spatial components of all of those objects simultaneously.

5.3.4 The User View

A stylized picture of the MPI screen (taken from our current implementation) is shown in figure 5-5. While the specific layout may change, the individual windows on the screen must support the following functions;

- interacting with the data source (GINDB)
- constructing individual overlays
- building aggregates from overlays
- manipulating the order of overlays within aggregates
- displaying aggregates

In our case, there are 7 windows. Reading left to right and top to bottom, they are; a color display of the current aggregate, a listing of data sources (GINDB responses) which the user can name, a display of the current data source, a list of all the overlays that have been defined, a listing of the overlays in the working aggregate (at the moment, only one aggregate is allowed), a palette for selecting overlay colors and modes (patterns), and a lisp interaction window which allows the TAPS user to converse with GINDB. The slides in Appendix A give color examples of TAPS screens.

The intent of this organization is to promote a work station format, in which

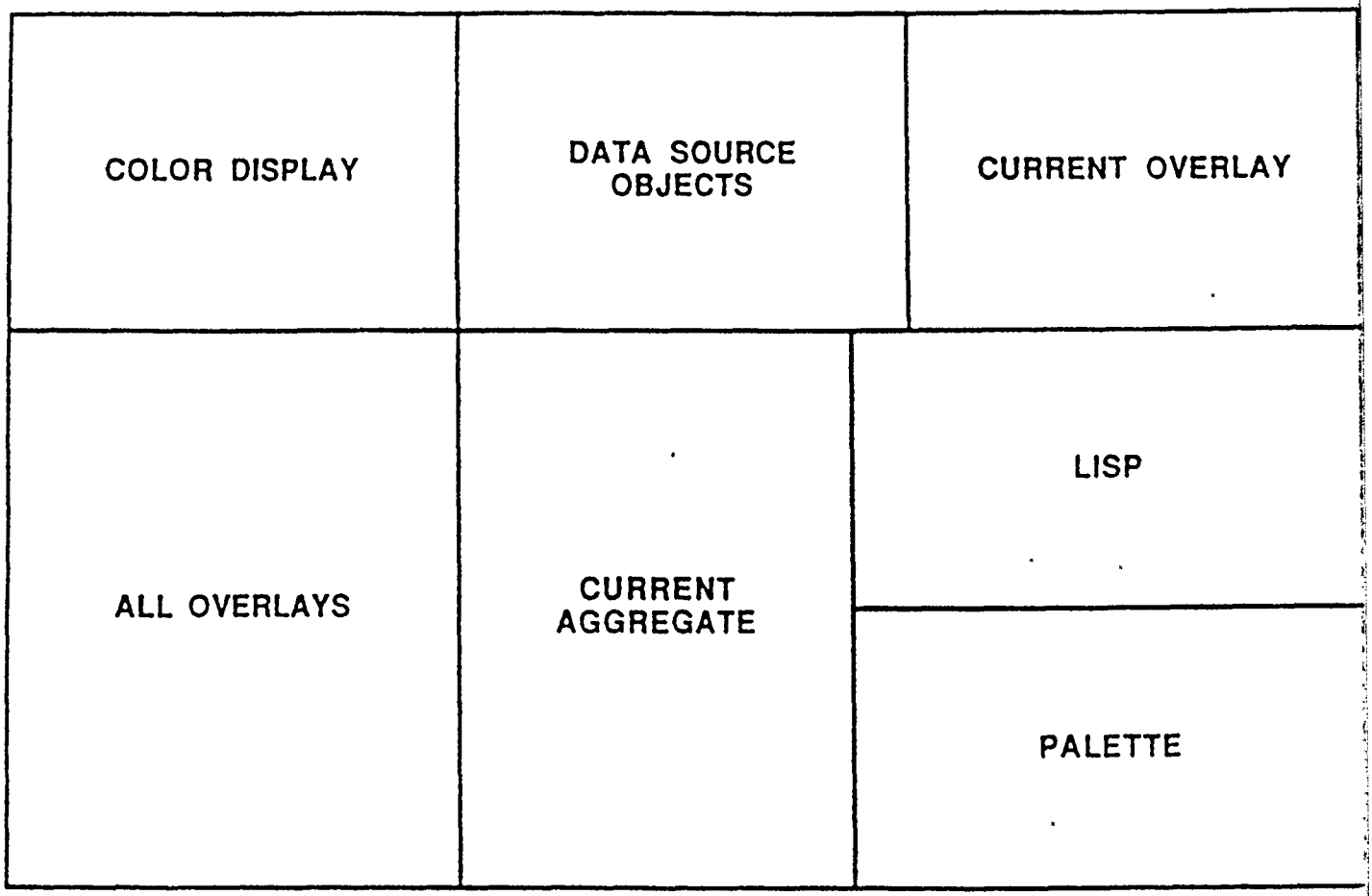


Figure 5-5: MPI Screen Image

overlays are produced, stored, and then displayed in different combinations to generate the information the user wants to see. To support a fluid interaction style, the objects in the windows are mouseable, and we have attached the major system operations (equivalent to operators on the data abstractions described above) to pull-down menus on the various windows. So for example, the user can mouse a color from the palette or a pattern, and then touch the icon for an overlay, and the overlay will be altered to display in that fashion (note that the icon encodes the color and pattern which will be employed). Similarly, the user can mouse the edges of the current aggregate window to obtain a menu for aggregate operations, and then select *reorder*, *delete overlay* or *display*, etc. Other miscellaneous features allow the user to name overlays, mouse on overlays and have their data source displayed (GINDB queries), and as a special interface with the GINDB interaction window, to mouse on overlays and have the (unsampled) bitmap they contain entered as region constants into a query under construction. This last feature was included in response to our observation that the user frequently wants to compute the intersections, unions and differences (etc.) of overlay panes. When the operation is generalized, the user will be able to take subsets individual panes, or regions from the color display, and input them into GINDB as a data source for further processing.

As our implementation progresses, we expect to add a new layer of TAPS specific mechanisms for automatically selecting certain colors or patterns, and for controlling screen clutter and overlay order based on a knowledge of the data involved. Several examples come to mind; water should always default to be displayed in blue, enemy motions in red, tactical objectives might always be shown as *bordered* areas, and linear features should always occupy overlays external to area data so that they are not obscured. In addition, text will eventually have to be treated specially so that it is never interpreted as a terrain object, either in extraction of data for GINDB or when describing the contents of the current screen.

5.3.5 Implementation Details

The following subsections discuss particulars of the current MPI implementation, as well as some modifications we plan to incorporate. In specific, we cover the algorithm and data structures used to produce multiple overlays, the implementation of specific overlay modes, and one alternate strategy (not employed) for the overlay process. This discussion is somewhat detailed; it is provided for those readers who are interacting with the MPI code, or for those whose curiosity knows no bounds.

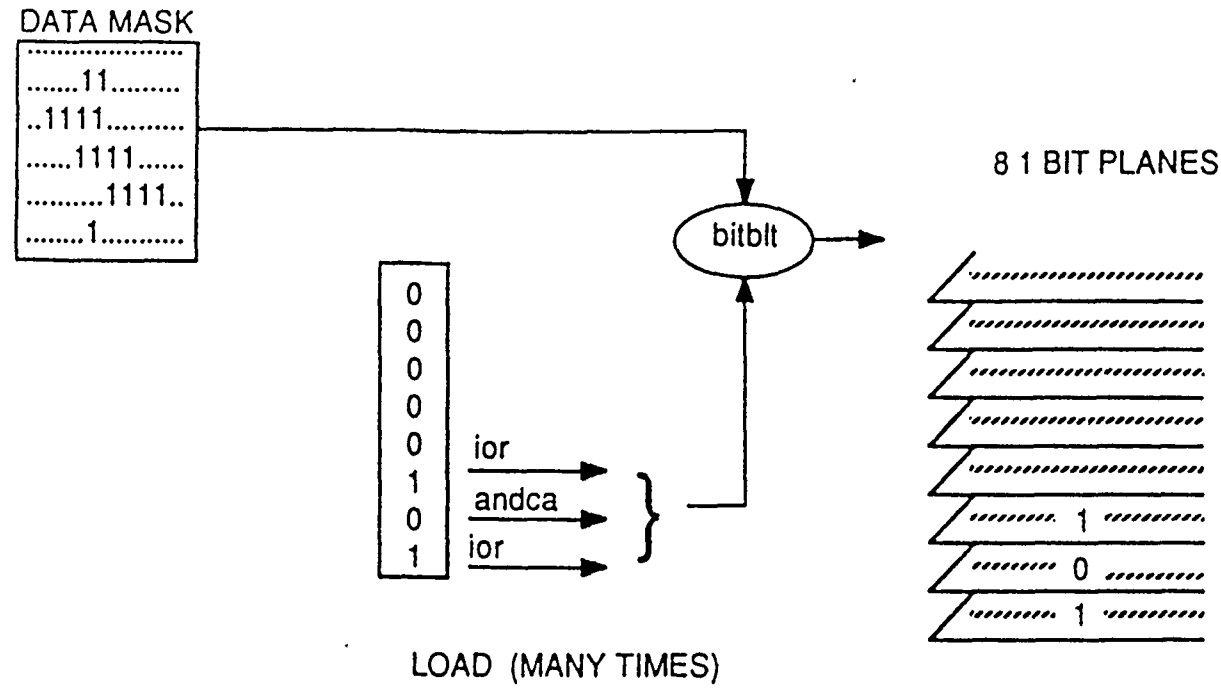
5.3.5.1 The algorithm for computing multiple overlays

From a graphics perspective, the problem in computing overlays is to take a 1 bit wide feature mask together with a number representing a color, and write that number into an 8 bit array (the color screen memory) at every x,y position where the feature mask shows data as opposed to background. This operation has to be done repetitively (up to 30 times) and efficiently if the MPI is to be a practical tool.

The naive method of handling this problem is to loop over the bits in the mask and bytes in the screen array, and write the appropriate number in the corresponding place. However, given that there are 262,144 elements in either array, the fastest loop we could write that treated each element individually took 4 seconds to execute. This would make computing 30 overlays a 2 minute process.

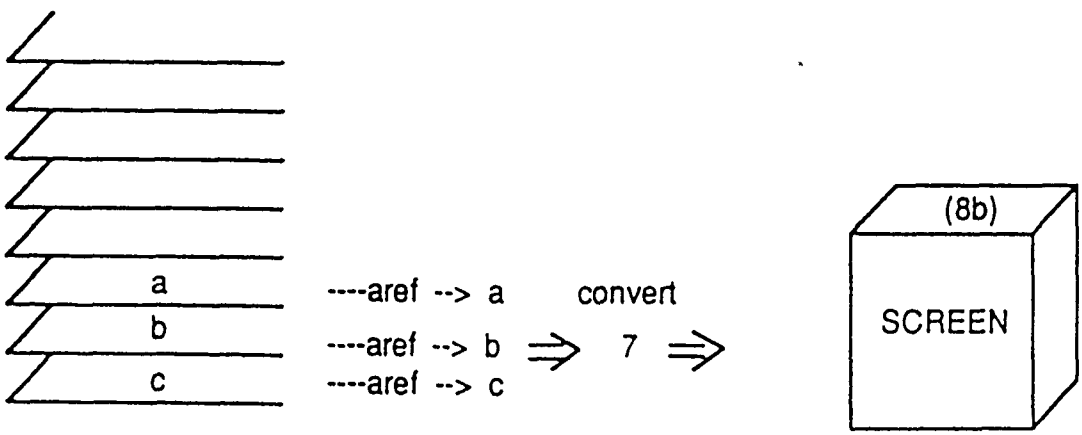
Our solution is shown in figure 5-6. The critical feature in this diagram is the data structure called the *plane cube* which is composed of eight, 1 bit wide, 512 x 512 planes which represent parallel slices through the bytes of the color screen array. This structure is used as an intermediate representation; once loaded with the appropriate color data, the contents of the plane cube are copied into the actual color screen memory.

The advantage of this approach is that the act of processing overlays into the plane cube is exceptionally fast. The task of dumping the plane cube (compacting 8 one bit arrays into one 8 bit wide array) is comparatively slow, *but it only has to be executed once* per aggregate display.



LOAD (MANY TIMES)

- TIME PROPORTIONAL TO # SIGNIFICANT BITS
  (5 BITS X 30 OVERLAYS --> .75 SECOND)



DUMP (ONCE)

- TIME PROPORTIONAL TO # arefs (TIGHT LOOP)
- EXPECTED = 2-4.5s

Figure 5-6: Use of the Plane Cube in processing overlays

The mechanism is as follows (see figure). Assume that the input feature mask is a bit array of all ones (feature everywhere, with no background), and that the number representing the color (say green) is decimal 2. Under these conditions, the task is to set the second plane in the plane cube to all ones, and to zero all other planes so only green will show. (A vertical slice through the plane cube encodes the number that will eventually reside in color memory. Only decimal 2's can be present above any point.) If the color number were decimal 3, then the first and second bit planes would have to become images of the input, and the third through eighth planes cleared instead. If we alter the feature mask so that it is not dense, the only difference is that certain vertical slices in the plane-cube should remain undisturbed, meaning the previous color, if any, should be allowed to show through.

In summary, the plane cube loading task is to take every place where the data mask shows feature, and set the vertical slice over that position in the cube to the color number (represented in binary form). This will require setting some bits and clearing others. Our observation is that this effect can be accomplished through a sequence of *bitblt* operations; if the color bit is a one, bitblt with Inclusive-Or will set that layer of the plane cube appropriately, leaving other 1's and 0's that might be present in background areas undisturbed. If the color bit is a zero, use of bitblt with *andca* is required. The truth table for *andca* (shown below) will turn all feature mask 1's into 0's, and will follow the prior value in that layer of the cube whenever the feature is 0. (The Symbolics supports all sixteen possible arguments to bitblt for comparing source bit, destination bit, and producing the binary result).

An efficient mechanism for dumping the plane cube data into one 8 bit array is described below:

1. Maintain a 1 bit wide mask denoting all places in the cube where any feature (of whatever color is present).

2. Iterate over this mask treating it as a linear array of 32 bit words.

3. Iteratively process the leading bit of the mask word, and if it is a one, assemble whatever color number is present at the corresponding x,y position

Table 5-1: The truth table for *andca*

Feature	Prior value 0	Prior value 1
0	0	1
1	0	0

of the cube into a single byte, which is then written (see below) into the screen array.

4. If at any time the mask word becomes zero (a single instruction test), none of the remaining bits are considered and processing moves to the next mask word.

5. When loading the color screen array, write it in 32 bit words vs. one byte at a time. If this is done, successive color bytes are loaded into a temporary register until four bytes are assembled.

In all of the above, array registers are used to speed up access times, fast memory (in this case, the control stack which is resident in the Symbolics processor as opposed to core) is used to store temporary variables (such as the 32 bit data mask), and feature arrays are wired down in core to prevent swapping delays. Some additional optimizations are actually available. For example, by allocating indices into the color table sequentially we can limit the number of planes in the plane cube which are ever occupied. This optimization has no great effect during overlay generation (btblts are essentially free), but it saves a large number of array references when dumping the plane cube to the color screen. All of these (somewhat Herculean) methods are required to obtain the necessary display times.

We have implemented the individual portions of this algorithm and are

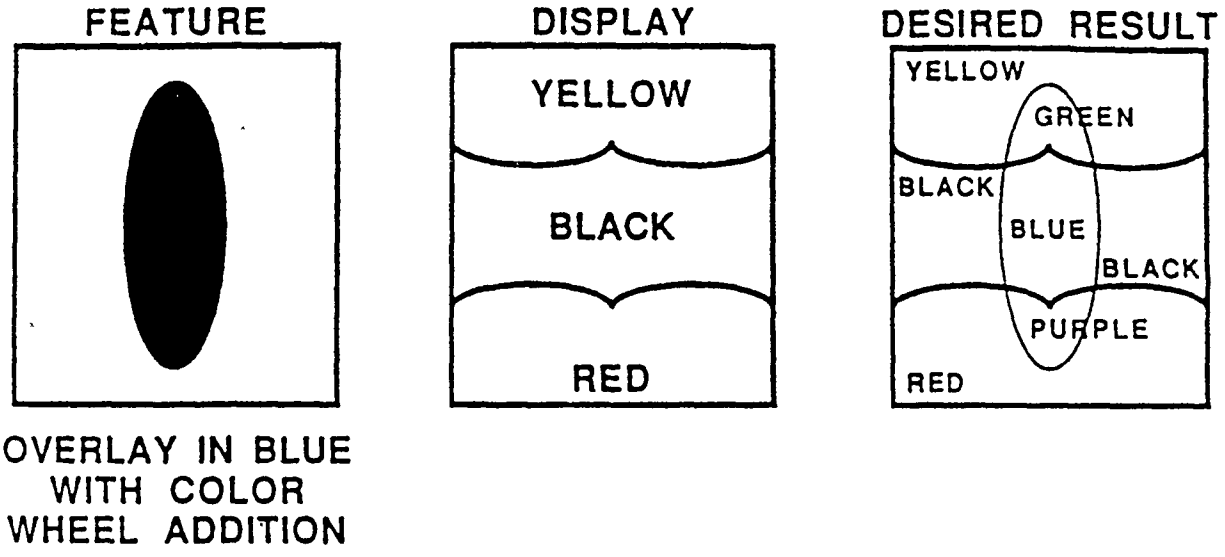
incorporating it into the MPI. Our current implementation employs a somewhat less optimized approach.

5.3.5.2 Implementations for the Translucency Operator

Recalling the discussion of overlay modes above, the purpose of the translucency operator is to allow the user to say, "overlay feature X on top of a multi-feature background, and show the overlap by tinting the background towards color C". A more general expression allows the result color to be computed as an arbitrary function of the inputs, for example, by color wheel addition. Since translucency is not a unary predicate on overlays, it does not fit in conveniently with the procedures we have expressed so far. As a result, we have had to consider several special case mechanisms, each of which is described below.

Figure 5-7 defines a simple translucency problem which is helpful in the following discussion. We refer to the binary feature (the source of the overlay data) as the *source* or the *feature*. The display (and the data it contains) are referred to as the *destination*.

In the first method, translucency is turned into a table lookup by a rather space inefficient mechanism. See figure 5-8. Here, we take advantage of the fact that we know the set of color indices which have been allocated for the display, and we compute the set of possible results of overlaying the translucent pane before any pixel operations occur. (If the data in the translucent pane happens to overlap all feature types present on the screen, there will be twice as many colors in the resulting display.) This information is stored in a table which associates the source color (which is either background or blue in the example) and the destination color (up to 256 possible) with the desired translucency result. As an efficiency trick, we make the table 256*256 elements long and index it by a number which is computed by concatenating the source and destination color indices. The array will be extremely sparse, but the index calculation and lookup can be accomplished in a few instructions, hence the net access speed is apparently quite high.



TRANSLUCENCY MAPPING
(FEATURE X DISPLAY ⟶ RESULT)

-- , BLACK ⟶ BLACK (BACKGROUND)

-- , YELLOW ⟶ YELLOW

-- , RED ⟶ RED

BLUE, BLACK ⟶ BLUE

BLUE, YELLOW ⟶ GREEN

BLUE, RED ⟶ PURPLE

Figure 5-7: A Simple Translucency Problem

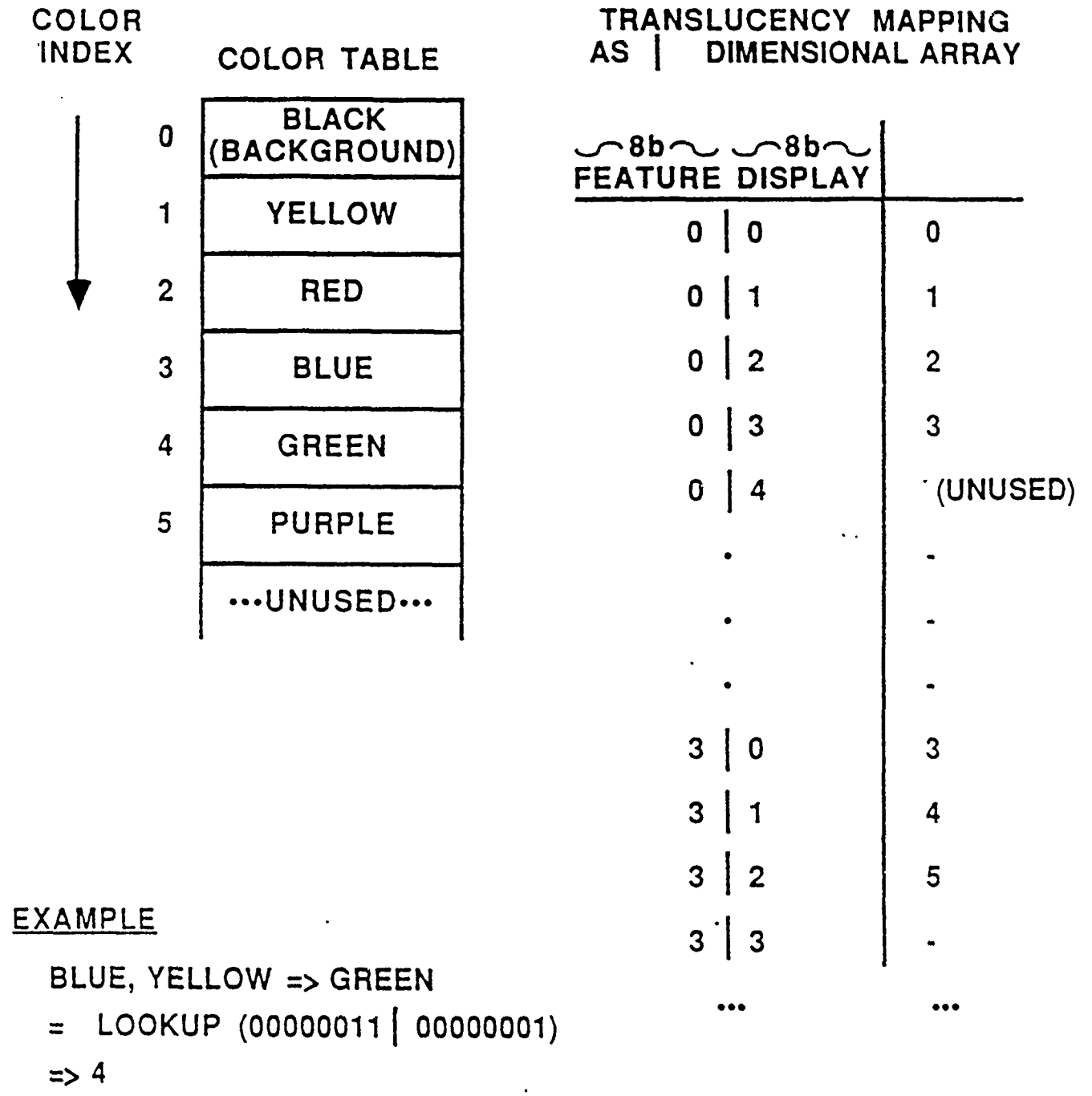


Figure 5-8: An implementation of translucency

This method is appealing, although it has several drawbacks. First, it turns out to be less efficient than desired because it requires accessing the binary feature map and the color screen array in the process of computing a single overlay. As discussed earlier, this results in computation times that are significantly higher than direct use of bitblt on the plane-cube (we have timed this translucency method at 2.6 seconds for our 512 by 512 arrays). Second, it is wasteful of color resource, in the sense that many colors are allocated which may or may not be used since only some feature/background overlaps occur.

Our second approach addresses both these issues. See figure 5-9. Here, we reserve one layer of the plane cube (say the one representing the highest order bit in each byte) for holding a translucent overlay. (By dedicating one entire bit plane to this purpose, half of the available color space becomes unavailable.) Next, we alter the color table to reflect all possible feature combinations as before, but we do this by duplicating entries for the existing indices with the leading bit turned on, and assigning those indices to the appropriate color.

When we set up the color table in this fashion, the desired effect of adding a translucent overlay is precompiled into the color table. We can now load the translucent overlay into the highest order bit plane of the plane cube (a single bitblt operation), and the chosen colors will appear *without having to reference the data in the screen array.* This translates into a marked gain in efficiency, but at the cost of half of the available color resource. Furthermore, it appears that only one translucent overlay can ever be computed, since that highest order bit plane is now occupied.

Our solution to this problem is to repack the plane cube in an off-line manner. That is, after the display has been produced we determine the new colors which were actually created (not all overlaps will occur) and move them to lower addresses in the color table (adjusting the color indices in the plane cube accordingly). We can then free up the color indices which were not required. If less than 128 colors remain on the screen, this once again frees the highest order layer of the plane cube, allowing

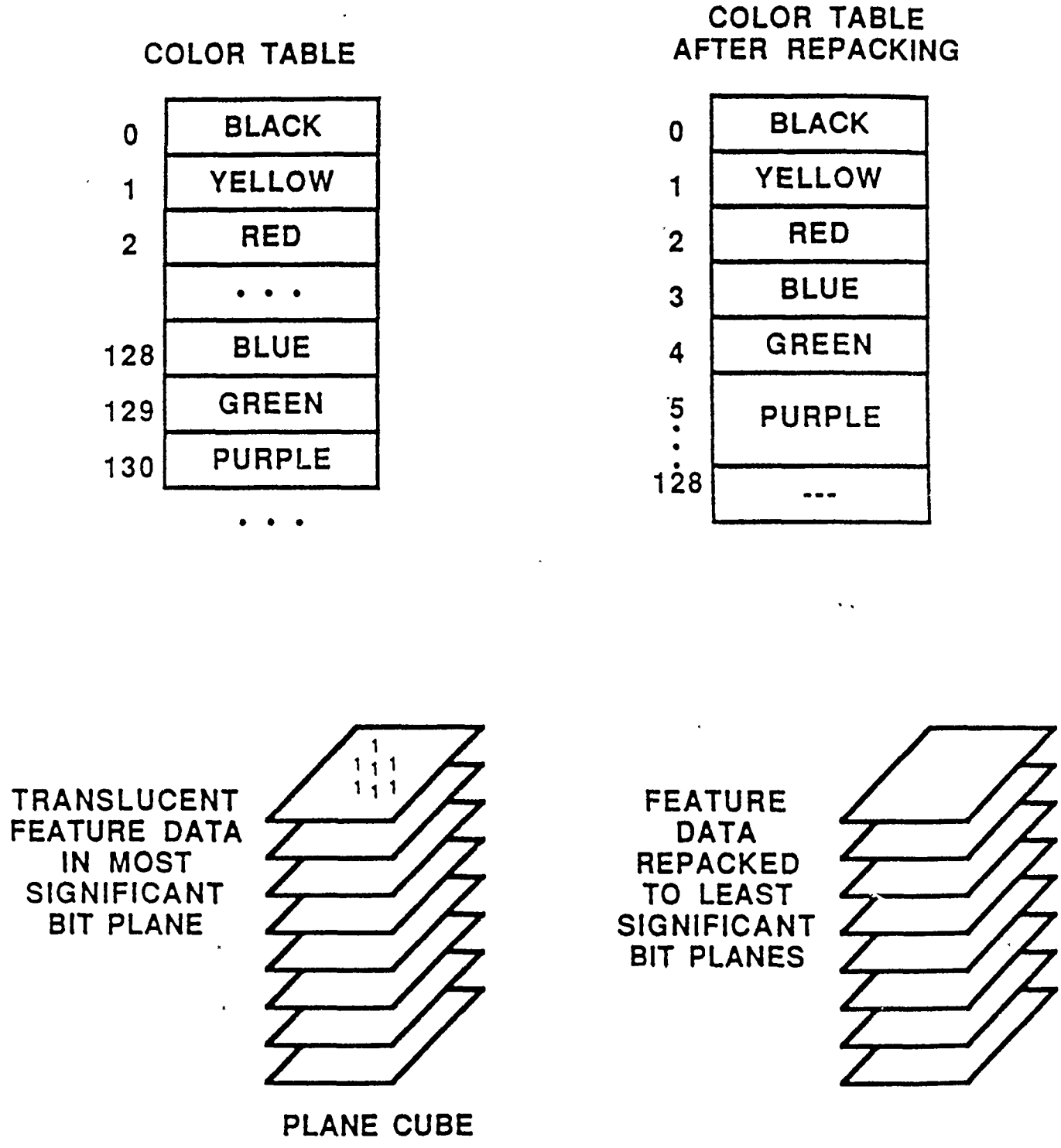


Figure 5-9: A second implementation of translucency

translucency to be reapplied. We have not implemented this function, and we do not expect it to be efficient in any way. However, it can be performed while the system is idle and waiting for new input from the user.

The third method of implementing translucency is to change the semantics of the operation to be "overlay feature X on feature Y and show the overlap in color C". Here, the inputs are both binary images, and the result is a single color, hence the computation is vastly simplified. It can be implemented as follows:

1. Compute the intersection of the two source features.

2. Define an overlay for the result with color C and solid overwrite mode.

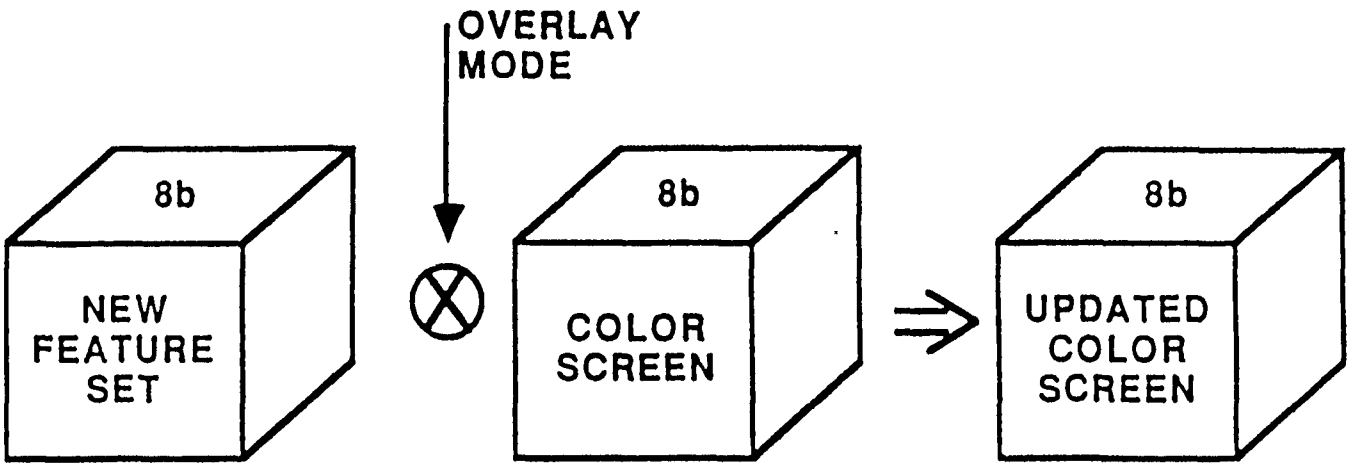
3. Place this overlay outside feature X and feature Y in the aggregate, and produce the display as before.

Since intersections can be computed via bitblt, this operation will require negligible execution time.

5.3.5.? An alternate overlay mechanism

From an I/O perspective, the mechanism above inputs a 1 bit array and an 8 bit array and produces an 8 bit array as output. An alternate approach to the overlay metaphor is to take two 8 bit wide arrays as input (a multi-feature source and a multi-feature destination - the display), and produce a new 8 bit display. See figure 5-10. Given that the source data in CATTS tends to come in multi-attribute fields (e.g., one field encoding six different road types), there is some justification from an applications viewpoint for adopting this metaphor.

In this view the semantics of combination is isomorphic to what has been described above. Overlay modes are still implemented as sampling followed by overwriting (except that multiple feature types are treated simultaneously), and the translucency operator now has to admit the possibility of producing n*m new colors (for n source and m preexisting destination feature types) instead of just 2*m. Both the feature and destination arrays are allowed to contain background areas, which means



e.g.,

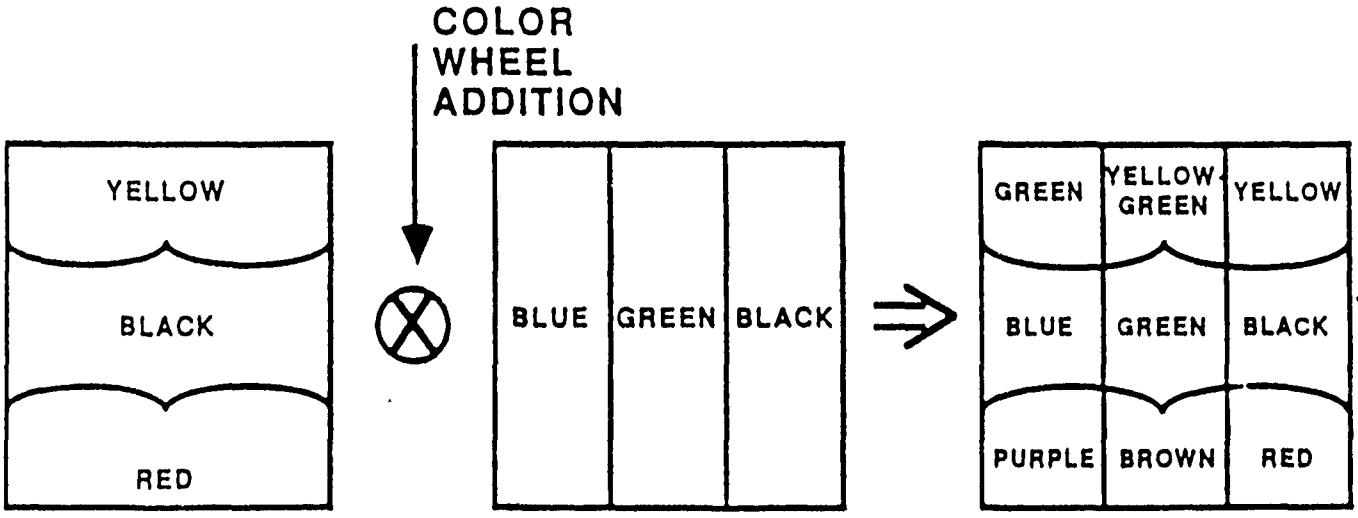


Figure 5-10: An Alternate Overlay Metaphor

that the overlay process wants to copy *bytes* in the input to bytes in the destination, except when the source byte is 0 (representing background).

It is tempting to implement this operation through an analog to bitblt which we might call *byteblt*. It is even worth hoping that there is microcode support for such an operation. It turns out that there is not, but worse, no reasonable building blocks are available either. The closest seeming function to work from is *copy-array-contents*, which is expressed in terms of bitblt, which in turn supports only bit-wise comparison operators. There is never a moment when an entire byte is available for testing, such that it can be copied to the destination or ignored as a whole.

We have written code which implements this metaphor (including the translucency operator, which was translated into an extremely efficient table lookup), but were only able to achieve 4-5 second overlay times. This in part stems from the lack of support, but also from the volume of data; each multi-feature array is 8 times the size of the dataset manipulated when considering binary feature images. Even copy-array-contents on a 512 x 512 byte array is comparatively slow; it requires over 2 seconds.

After exploring this path, our intuition was that it was less flexible and less easily implemented than the overlay metaphor we chose. In addition, we suspected that more information would have to be manipulated to produce the same display, even though fewer overlays (with more features apiece) would be involved. This would indicate that equivalent processing times could never be achieved.

It is possible that this second metaphor will be incorporated into TAPS as a technique for overlaying aggregates as opposed to individual panes.

5.4 The Geographically Intelligent Database (GINDB)

GINDB is a query language for accessing and manipulating terrain features (in our case, information obtained from the digitized CATTS database although attachment to CATTS is not critical). GINDB represents structured terrain objects and the properties

of terrain with relations, and provides a query language over them (similar to the "non-procedural" query languages of the Ingres project [Ullman 82]) that makes it possible to retrieve objects based on their descriptions, and to combine and construct regions via any of the algebraic set manipulation operators. A mechanism for incorporating n-ary relations between objects has also been included, and the system as a whole has been implemented in lisp on the Symbolics 3600. It is quite efficient; typical queries require on the order of a few seconds to compute.

GINDB should be considered as an experimental vehicle which will stabilize after various representational possibilities have been explored. However, we have made a serious attempt to support the functionality required by a variety of terrain reasoning scenarios (suggested by applications at ADS to terrain analysis and military planning, processing of tactical imagery, robot vision, and automated route planning), meaning that the implementation functions as a prototype for a generally applicable terrain reasoning tool.

The following sections describe the terrain storage and retrieval problem, our major design decisions, and then the semantics of the GINDB query language, including information about the scope of the implementation and the underlying representations. We conclude with an appendix containing output from several demonstration senarios.

5.4.1 The Problem

The problem addressed by GINDB can be expressed as follows; given a set of terrain features in the environment, provide a facility for retrieving them according to their properties, and for combining them according to standard set manipulation procedures. However, by looking at more detailed BCA scenarios, and by projecting towards future needs (for BCA and other terrain reasoning efforts), we can impose the following set of additional requirements;

- it must be possible to set a geographical bound on the search query
- a given search query must allow arbitrary n-ary predicates to be applied to the terrain regions under consideration

- the underlying representation must allow a declarative expression of n-ary relations

- the system must support structured terrain objects which contain an arbitrary amount of attached information (e.g., properties, attached procedures, and spatial extent)

- the query answering process must be computationally efficient

- the representation must support point, line and region features

- the query language must not commit to any single representation for space. In particular, it must allow for the possibility of simultaneously employing bitmaps, line segments, quad trees, k-d trees, and polygons (see [Shapiro 85a] for details).

5.4.2 Approach

The main decision involved in our work on GINDB has been to adopt a relational database approach. This implies a commitment to a particular style of query language and mechanism for interpreting database requests, and a decision to represent the component features of objects as relations. As a result, we have obtained excellent retrieval efficiency and the generality associated with the relational mechanism, but at the loss of the declarative simplicity provided by the more semantic-net oriented view.

To set up the contrast, semantic-net approaches represent objects declaratively as single entities which can then have any number of attached parts. These objects are then related by named links which provide a vocabulary for the interactions which the system can represent. For example, the ISA link gives rise to a hierarchical class structure, which in turn supports the inheritance operations that are now widely known. As a second example, we proposed a net formalism in [Shapiro 85a] that employed links for spatial relations (containment, adjacency, enclosure, etc.) to simplify the computation of set operations on geographical regions. In general, semantic nets will have interpreters that supply a variety of operations to compute over the known links; examples are demon invocation (in net languages such as FRL), object matching (in all recognition scenarios), and representation transformation (in conceptual dependency parsing and response generation).

In the relational database view, the representation for an object is less declarative and more procedural (or the concept of an object as a whole is less well defined), meaning that specific operations require more search. So for example, if you are given an object ID in a semantic net, you have a package which contains the object parts (they are primitively available). In a relational database, the object name allows you to look up the object in a table containing all objects in the world. The answer to that search is the relation which contains the desired information. This distinction persists when considering relations between objects. In a semantic net, given an object you can find the thing which it is "a kind of" by following explicit pointer links. In the relational view, there would be a table of ISA relations, and the process would require a search through all known relations of that kind.

The value of the relational model comes from the generality with which it specifies retrievals. In essence, by breaking apart objects and forming relations, any set of properties can be used as indices to locate relevant features. Said in a different way, if the purpose in an application is to find features which obey a (potentially complicated) description, the relational view provides a natural format for making that occur. The BCA application fits this criterion very well.

In summary, our reasons for chosing the relational approach are as follows:

- The BCA project is concerned both with retrieving and constructing new objects by the fact that they exhibit certain properties. The exact set of those properties is unknown at this time, which argues for a more homogeneous approach which allows arbitrary descriptions to be combined.

- At the current time, the focus is on the storage, retrieval and manipulation of regions of space, as opposed to reasoning about the objects which occupy the terrain. Type hierarchies, inheritance, and similar domain structuring mechanisms are not yet required.

- The primary goal has been to produce a flexible query language in a short period of time; the relational approach has simplified that task, even though it may not be a unique solution.

Our second major design decision has come from examining the types of objects in

the BCA domain. In particular, we have separated the notion of a structured terrain object with many attached properties (called a "Thing") from the concept of a region of space which is not allowed to carry any descriptive information (called a "Spatial Region" or "Region"). The intent here is to capture the notion that some queries result in classifications of large terrain areas (as when forested Regions are overlayed on hilly Regions), while others pertain to specific known Things (like finding the hilltops which have observability over forest #12). From a computational perspective, the Things can be expected to identify reasonably cohesive regions of space, whereas areas built from combining arbitrary features cannot. This distinction has provided several simplifications in the query language, and it may lead to some easy representation choices when multiple representations for space are entertained.

The last critical design decision has been to hide the actual representation of terrain regions from the user, and to write the query language such that it admits any number of potential representations. The reasons for this become clear when one looks at the terrain representations that naturally come to mind; none is optimal for computing set operations over all types of terrain regions. Several in combination might be desired.

To be specific, quad trees have significant space requirements and set up times, but generally provide quick manipulation procedures. However, both the storage and computational efficiency properties break down if the underlying regions are not significantly cohesive. In addition, it is cumbersome to express individual objects (with many attached properties) given a language that provides only binary exclusion primitives. Polygonal representations are excellent for establishing objecthood and for expressing relations between features, but they give rise to awkward implementations of the set manipulation operations. This is easy to perceive when considering Region features of the kind described above. They will tend to have complex shapes, be non-contiguous, non-convex and include holes. This leads to manipulation operations as bad as n**4 in the number of vertices. (The system described in the BCA annual report was an attempt at addressing this efficiency issue.) Bitmap representations can be used to

represent point, line and region features, and the set manipulation operations are very easy to express. They run in constant time (given by the maximum region size), and are supported by all bitmap display processors that include a "btblt" function. However, the representation is certainly not space efficient, and is less optimal than the others in situations where they excel.

As a result, we have defined a query interface which is independent of the actual representation chosen for physical space, and we have been thinking in terms of supporting many representations simultaneously. For the current implementation however, we avoided dealing with the attendant conversion problems and experimented with only the bitmap view.

5.4.3 Semantics of the Query Language

5.4.3.1 Definitions

A WOREL (WORld ELement) is a unit square in the grid which represents the world of the database system. The world is a rectangular grid of worels where each worel is uniquely identified by its coordinates.

A REGION is a set of worels.

AN OBJECT is a distinct entity in the world. Generally only something which is uniformly best thought of as a single object, such as a hilltop, is denoted as an object. Most objects probably have an associated region called a spatial component.

A RELATION is an n-ary named relationship between objects.

When finished, the capabilities of the system are to be as follows:

LS 11 OCT 90)

AMSEL-RD-C3-EM-F (AMSEL-TMDE-~~MG~~/6 Sep 90) (70-17b) 1st End J. Van Savage/ldw/42503

SUBJECT: Procurement Data Call ~~Meeting for~~ Radio Frequency Power Test Set, ~~AN/USM-~~ For the EOP BSTF-EOB

CHIEF, ELECTROMAGNETIC ENVIRONMENTS DIVISION 22 OCT 1990

FOR AMSEL-TMDE-~~MG~~ (~~M. BRATSTEIN~~) LS A. PEARSON

1. The EMI PDP inputs are as follows:

a. Specification of Inputs (Encl 2).

b. Statement of Work (Encl 1).

c. CDRL Justification (Encl 3).

d. DD Form 1423 (Encl 4).

e. Document Summary List (Encl 5).

2. The EMI CDRL's have been reviewed and found to be the minimum essential requirements.

3. POC for this action is Mr. John Van Savage, x42503.

4. CECOM Bottom Line: THE SOLDIER.

5 Encls
as

WARREN A. KESSELMAN
Chief, Electromagnetic Environments Division
CECOM Center for C³ Systems

DISTRIBUTION:
CDRL-RD-GT
AMSEL-LC-ME-MT
AMSEL-LC-ME-PMT
AMSEL-LC-ME-T
AMSEL-LC-ME-I
AMSEL-LC-ME-V
AMSEL-PA-MT
AMSEL-RD-C3-EM-M
AMSEL-RD-C3-TD-T
AMSEL-SF-S
AMSEL-TMD-M
AMSEL-TMDE-MB
AMSEL-TMDE-MG

EM-F (Van Savage)

define regions
update regions
delete regions

define objects
insert objects
update objects
delete objects

define relations
assert facts (that a particular relation holds between n objects)
unassert facts

query regions and objects using a "non-procedural" query language
(discussed in the the following sections)

At the current time, it is possible to define regions, to define and insert objects (in an off-line manner), and to build an extensive set of non-procedural queries.

Additional hybrid objects might be useful. For example, (taking a mobile robot context) a *possible object* might be stored having a spatial component represented by a pie slice emanating from the point from which it was sighted. At some later time, when the actual location and shape of the object had been found, the original *possible object* could be deleted, and the identified object inserted. The key point here is to introduce the notion of a temporary, hypothetical object into the database.

5.4.3.2 Development of the query language

In order to explain the semantics of the proposed query language, we will develop several example queries that operate over a few normalized relational tables representing terrain. We use the QUEL language (developed in the INGRES project) as a point of departure, and show the transformation of those queries into the lisp-like syntax of the implemented language.

Initially we start with the two relations in figure 5-11, and want to answer the following query:

Relation: properties of worels

x	y	property
35	511	forested
36	510	forested
37	510	forested
12	256	hilly
15	15	object1
15	16	object1
16	16	object1

Relation: objects in the world

objectid	kind	otherprop
object1	plateau	otherval
object2	hill	otherval2

Figure 5-11: Two relations representing the world

Query 1: Give me all plateaux which are partially in a forested, hilly area.

In Quel this becomes:

```
range of w1, w2, w3 is worels
range of t is objects

retrieve (t.objectid)
where t.kind = "plateau"
  and t.objectid = w1.property
  and w2.property = "forested"
  and w1.x = w2.x
  and w1.y = w2.y
  and w3.property = "hilly"
  and w1.x = w3.x
  and w1.y = w3.y
```

We can simplify the query a little by making a distinction between objects and regions, that is between objects which have other properties besides being aggregations of worels, and objects which are only aggregations of worels. The relation forested now represents the region which is the set of worels which are forested. Figure 5-12 shows this schema. Query 1 can now be simplified to:

Relation: forested

x	y
35	511
36	510
37	510

Relation: hilly

x	y
12	256

Relation: spatial components of objects in the world

x	y	objectid
15	15	object1
15	16	object1
16	16	object1

Relation: objects in the world

objectid	kind	otherprop
object1	plateau	otherval
object2	hill	otherval2

Figure 5-12: N relations representing the world

```
range of f is forested
range of h is hilly
range of s is spatial-components
range of t is objects

retrieve (t.objectid)
where t.kind = "plateau"
      and t.objectid = s.objectid
      and s.x = f.x and s.y = f.y
      and s.x = h.x and s.y = h.y
```

But suppose we want to ask the following question:

Query 2: Give me all plateaux which are entirely in forested, hilly areas.

This has a much more complicated translation into QUEL:

```
range of f is forested
range of h is hilly
range of s is spatial-components
range of t is objects

retrieve (t.objectid)
where t.kind = "plateau"
  and t.objectid = s.objectid
  and count(s.x, s.y where s.objectid = t.objectid)
     = count(s.x, s.y where s.objectid = t.objectid
                  and s.x = h.x and s.y = h.y
                  and s.x = f.x and s.y = f.y)
```

What is wrong here? Talking about single worels seems to be unintuitive -- let's switch over to talking about sets of worels by introducing the following relations and definitions.

define setofpairs(s.x, s.y) to be all pairs <x, y> from relation s

define UNION, INTERSECTION, and EMPTY-SET as usual

Queries 1 and 2 can now be reformulated as follows:

Query 1:

```
retrieve (t.objectid)
where t.kind = "plateau"
  and t.objectid = s.objectid
  and INTERSECTION(setofpairs(s.x, s.y),
                              setofpairs(f.x, f.y),
                              setofpairs(h.x, h.y))
        <> EMPTY-SET
```

Query 2:

```
retrieve (t.objectid)
where t.kind = "plateau"
  and t.objectid = s.objectid and
  INTERSECTION(setofpairs(s.x, s.y),
               setofpairs(f.x, f.y),
               setofpairs(h.x, h.y))
        = setofpairs(s.x, s.y)
```

We can simplify still more by substituting the names of the relations f and h

range over for the expressions "setofpairs(f.x, f.y)" and "setofpairs(h.x. h.y)". and the function spatial-component(t) for the expression "setofpairs(s.x x.y) where t.objectid = s.objectid". Query 2 then becomes:

```
retrieve (t.objectid)
where t.kind = "plateau" and
INTERSECTION(forested,
             hilly,
   spatial-component(t))
   = spatial-component(t)
```

Lastly we further separate objects from regions and introduce the predicates in-region, and entirely-in-region.

```
Query 1:

retrieve (t.objectid)
where t.kind = "plateau" and
  in-region(t, region-that-is(forested and hilly))
Query 2:

retrieve (t.objectid)
where t.kind = "plateau" and
  entirely-in-region(t, region-that-is(forested and hilly))
```

The following question requests the map, or set of worels that meet a certain property;

Query 3 Show me the region which is forested and hilly.

When translated into QUEL this becomes:

```
retrieve (w1.x, w1.y)
where w1.property = "forested"
  and w2.property = "hilly"
  and w1.x = w2.x
  and w1.y = w2.y
```

Which, using the above simplifications, can be translated to something like:

```
retrieve-region-that-is(forested
                        and hilly)
```

A similar, but more complicated example is to ask the question:

Query 4 Show me the region which is part of some plateau and forested.

This becomes:

```
range of t is objects

retrieve-region-that-is(forested and spatial-component(t))
where t.kind = "plateau"
```

Note: by translation back to QUEL and implicit existential quantification the query asks for all worels in the world which are parts of plateaux such that the places are also forested. If one asked:

Query 5: Show me the region composed of all plateaux which are somewhat forested.

The following query is the result:

```
retrieve-region-that-is (spatial-component(t))
where t.kind = "plateau"
  and in-region(t, region-that-is(forested))
```

Finally we present translations of all of these queries into the lisp syntax implemented within GINDB (note that the implementation uses the term "thing" everywhere where the term "object" might be expected, thus the top level retrieval function is "retrieve-thing", and not "retrieve-object"):

Query 1: Give me all plateaux which are partially in a forested, hilly area.

```
(retrieve-thing $t (and (plateau $t)
                        (in-region $t (and forested hilly))))
```

Query 2: Give me all plateaux which are entirely in forested, hilly areas.

```
(retrieve-thing $t (and (plateau $t)
                        (entirely-in-region $t (and forested hilly))))
```

Query 3 Show me the region which is forested and hilly.

```
(retrieve-region (and forested hilly))
```

Query 4 Show me the region which is part of some plateau and forested.

```
(retrieve-region (and forested (spatial-component $t))
  (plateau $t))
```

Query 5: Show me the region composed of all plateaux which are somewhat forested.

```
(retrieve-region (spatial-component $t)
  (and (plateau $t) (in-region $t  forested)))
```

5.4.4 Language Syntax

The following is a BNF syntax for the GINDB query language:

query -> retrieve-thing
| retrieve-region

retrieve-thing -> (**retrieve-thing** result-expr optional-thing-predicate)

retrieve-region -> (**retrieve-region** region-expr optional-thing-predicate)

result-expr -> thing-expr
| (thing-expr+)

thing-expr -> variable
| thing-constant

optional-thing-predicate -> /* the empty expression */
| thing-predicate

thing-predicate -> predicate-expr
| (**and** thing-predicate thing-predicate+)
| (**or** thing-predicate thing-predicate+)
| (**not** thing-predicate)

predicate-expr -> lisp-escape-expr
| in-region-expr
| (relation-name thing-expr+)

relation-name -> thing-type-constant
| thing-property-constant
| thing-relation-constant

lisp-escape-expr -> (lisp lisp-expr)

lisp-expr -> *general lisp expression, including query variables*

in-region-expr -> (**in-region** variable region-expr)
| (**entirely-in-region** variable region-expr)

distance -> number

```
region-expr -> region-name
             | parameterized-region-name
             | boundary-expr
             | computed-region-expr
             | computed-with-data-region-expr
             | thing-spatial-component-expr
             | region-constant
             | (and region-expr region-expr+)
             | (or region-expr region-expr+)
             | (xor region-expr region-expr+)
             | (equiv region-expr region-expr+)
             | (not region-expr)
parameterized-region-name -> (slope minslope maxslope)
boundary-expr -> (bounded lowx highx lowy highy)
computed-region-expr -> (in-direction ctrx ctry direction width radius)
computed-with-data-region-expr -> (nearby region-expr radius)
thing-spatial-component-expr -> (spatial-component thing-expr)
variable -> $name
region-constant -> (region region-handle)
thing-constant -> (thing thing-handle)
thing-type-constant -> hill
                     | plateau
thing-property-constant -> sandy
thing-relation-constant -> on-the-path-between
                         | connecting
region-name -> forested
             /* defined as (or coniferous deciduous) */
             | hilly
```

Most of this syntax can be understood by looking at the examples in the section entitled "example queries". However, it is worth giving a few notes here.

The language allows the user to escape into lisp in the middle of a query in order to apply tests to any set of things (objects) which are in the process of being examined. This is accomplished by use of the form (lisp lisp-expr) as above. For example, the predicate

```
(and (hill $x) (lisp (bigger $x 2)))
```

filters hill objects to see if they are larger than 2 square kilometers, where the "bigger" predicate is written in lisp and independently accesses object parameters.

It is also possible to pass regions and objects into the query language from lisp (to allow partial results to be stored in an application program). This is done through use of the forms:

```
(thing expr)
(region expr)
```

Where expr is a lisp expression that produces an object of the appropriate type. For example, the query

```
(retrieve-region (and (region old-data) forested))
```

and's the region stored in the lisp atom "old-data" with the forested region resident in GINDB.

As a final note, because of the distinction we have drawn between objects and regions, we think in terms of constructing regions and running predicates on objects. As a result, there are a variety of region expressions which perform the following functions;

- **nearby** computes the envelope of a given region, as in all worels within 3 worels of a road.
- **bounded** returns a rectangular region of the specified dimensions
- **in-direction** returns a pie-sliced region centered on the specified x-y coordinates, with a given angular dimension and range.
- **slope** computes the region consisting of all worels on slopes within the bounded degree range.

Thing predicates (see predicate-expr above) filter existing objects via tests written in lisp, predicates on an object's spatial component, or according to (see relation-name above) the type, properties, or presence of n-ary relations between an object and other objects in the environment. Examples here are the predicates

- **in-region** which restricts an object to have a spatial component which is partially in a designated region
- **entirely-in-region** which demands complete enclosure, and

- **hill** which determines if an object is of type hill

- **sandy** which limits an object to have the property of being sandy

- **on-the-path-between** which determines if a particular relation exists, namely that the spatial component of an object overlaps the line drawn between two others. This form can be implemented either by a built-in function, or by checking for the appropriate static links between component things. As the query language develops, it will be possible for the user to define new relations, and query them using this form.

Note: at the current time, N-ary thing predicates (such as on-the-path-between) and thing property constants (e.g., Sandy) are not implemented.

5.4.5 Representation and Data Structures

In the current implementation, regions (and spatial components) are represented by bitmaps (although the query language makes no commitment to that form). This approach was used both for simplicity and as an experiment as to the effectiveness of such a representation.

Objects are represented by Zeta-Lisp flavors, which are the basic mechanism for building structured objects in Zeta-lisp. They allow an arbitrary amount of attached data, procedural embedding, and provide standard operations for inheritance through a tangled class hierarchy. The spatial components of objects are also represented by bitmaps. It is an open issue whether distinct flavors are to be used for distinct kinds of object, or whether different properties of different kinds of objects will be handled by the lisp property list mechanism.

Static relations (which are currently not implemented) will have a number of representations. One to One relations will be represented by double linking. One to Many and Many to One will be represented by single links one way, and lists of links the other way. N-ary relations (with N greater than two) will be represented by more complex relation structures with appropriate links. The intent is to separate knowledge about the actual representation of objects from the query evaluation code, so that

different underlying structures can be added or substituted without upsetting the upper layers.

We have also introduced some special representations to optimize retrieval performance. In particular, since queries often request the set of objects which exist within a given region, we have implemented a pyramid/quadtree structure which provides the required spatial index. The presence of this structure also simplifies certain spatial operations on objects such as determining intersection and disjunction. It works as follows; each object is inserted once in the structure in the smallest square of the pyramid which completely encloses it. Given an object, the candidates for intersection must live in the same pyramid square, or in one of the squares which are enclosed within it. Similarly, the set of all objects which don't intersect a given object can be computed with reference to the pyramid (as opposed to running N intersections where N is the number of known objects). So far, the pyramidal index has provided excellent results, although an actual performance analysis might point to other, more efficient organizations.

The final data structure of significance in GINDB is the "data dictionary" which stores information about the regions, relations and object-types. This structure can be examined by looking at the lisp variables *Relations* and *Regions*. The sum total of objects in the environment can currently only be accessed through use of the query language.

5.4.6 Algorithms

The semantics of our query language can be derived from relational languages, so the same query processing algorithms are applicable. However with the addition of the region constructs and the main-memory nature of the data-base, we have the ability to add special forms of optimization based on the special representations we employ.

Our basic approach is to create an ad hoc lisp function that generates the results specified by the query. This method has several advantages;

1. The query is actually processed as a compiled procedure, rather than as an interpreted one.

2. The resulting function can be repeatedly executed on different data, thus avoiding the cost of planning an execution strategy.

3. Such a system may be easier to debug; since processing is partitioned into two phases (preprocessing and execution), errors are also partitioned. In addition, the result of preprocessing is an actual program which can be examined.

The outline of the algorithm used to generate the function is as follows:

First, convert query predicate to conjunctive normal form and introduce variables for n-ary relations and thing-constants. (Thing-constants are references to actual objects. Thing-constants are replaced by bound variables.) Then, recursively apply the following rules, choosing from the top following each successful application of a rule (these steps implement the query optimizations referenced above):

1. If there is a conjunct (disjunction) which has not already been satisfied in some way, and in which all variables have already been bound, generate code to test this conjunct. (an example would be if $x were bound to range over all objects in the world, and one conjunct were p($x), then code for testing p($x) could be generated).

2. If any region expressions are free-variable free, generate code for them. (examples would be forested, or (and forested (spatial-component $x)) where $x is bound).

3. If there are no free variables, generate an output expression. (an example is the query (retrieve-thing $x). After $x is bound to range over all objects in the world, all that is left to do is produce output, that is generate code to cons $x onto the output list).

4. If any connected (connected by n-ary relations) set of free-variables contains no output variables, generate code for them as an existential variable. (an example of this is (retrieve-thing $x (and (2-ary-relation $x $y) (hill $y) (house $x))) if $x has been bound to range over houses, then $y is disconnected from any output variable (since $x is bound and is the only output variable). Therefore $y can be treated as an existential predicate, and the query can be translated into the form: is there a $y such that it is a hill and 2-ary-relation($y, $x))

5. If any region expression contains free variables, and the free variables are disconnected from free variables in the rest of the query, build code for generating the region which represents the union of all bindings of the variables in the region expression. (an example of this is (retrieve-thing $x (in-region $x (spatial-component $y))) where $y is disconnected from any variable, and hence the region denoted by (spatial-component $y) can be constructed).

6. Choose a variable to iterate over. The choice is done heuristically based on the cardinalities of the sets the variables range over, and the ease of finding related objects. The iteration can be done in a number of ways. The broadest kind of iteration is iterating over all objects in the world. If a kind (or type) predicate is used ((hill $x)) and it is the only element of a conjunct, then only the kind need be iterated over (only hills need be considered). If the variable is bound in a region expression which appears as a single conjunct in conjunctive normal form ((in-region $x forested)), then the variable can be bound using the pyramid structure, only iterating over those buckets of the pyramid which overlap the specified region (forested). Important rules of thumb are: if a relation is many to one, iterate over the set of ones first; in an in-region expression, iterate over variables in the region expression before the object variable.

There are a few other interesting algorithms: the blurring function for the nearby region expression, and the test which determines if there is at least one bit set in a bit array.

The nearby region expression, given a region (bitmap) and a radius as input, is intended to return a region which has grown by the radius. Every pixel that was originally 1, should eventually be surrounded by a circle of 1 bits of the specified radius. This is done by first or'ing the original image on itself in a circle of radius 1, followed by or'ing the resulting image on itself in a circle of radius 3, and so on. Example applications of nearby are shown in the following chapter.

Given a bitarray, the "set bit function" (called "any" in the implementation) returns true if any bit is set, false otherwise. Obviously there should be a machine instruction to do this, but there isn't. Even an optimized loop through the array testing for set bits takes seconds. The solution to the problem is to logarithmically *or* the bit-array onto itself, effectively folding any set bits into an area which is tractable for

ordinary loops and bit tests. This makes efficient use of the btblt function which is supported in microcode.

5.4.7 Incorporating GINDB into TAPS

This section briefly discusses the additions we have made to GINDB in order to integrate it into the context of TAPS. Most of the changes were minor, having to do with the inclusion of new interface code, although several were substantive.

The following changes have been introduced:

- a capability to handle n-ary relations
- two new query forms for passing *region* and *thing* constants into GINDB
- an internal database for storing these newly produced items, and
- a new interface which allows GINDB to be invoked as a function, and which returns a *query response structure* (QRS) that encodes its results

Of the above, the capability to handle n-ary relations is the most significant (in terms of difficulty) since it required changes to many of the GINDB internals. The issue is that the prior version of GINDB only knew about the unary relation of object type while the bulk of the implementation was concerned with supporting the manipulation of spatial data. Relations were implemented as simple lists of elements, and member searches as simple enumerations. In contrast, N-ary relations are implemented as n-tuples; this requires alterations to GINDB's code generator to accommodate enumeration and selection of fields from those tuples.

Some operations such as logical negation also take on a new level of complexity when applied to n-ary relations. In the unary case, the expression

```
(not (bridge $b))
```

was implemented in one of two ways; if the object $b was already bound in the context of a larger query, then GINDB's coder produced a membership test for $b in the set of bridges. If $b was not bound, then GINDB enumerated the set of all objects

and filtered each for membership in the bridge relation as above. In the n-ary case, the expression

```
(not (bridge-over-stream $b $s))
```

can give rise to several meanings; one interpretation is to return the set of tuples

```
[bridges X streams]
     - <tuples from the bridge-over-stream relation>]
```

which requires enumerating the elements of the two unary relations and forming their cross product. Another interpretation is to form the set

```
(not <bridges from bridge-over-stream>)
     X (not <streams from bridge over stream>)
```

which is similar to the above though less permissive (see figure 5-13); it refuses to include tuples such as {BR4,ST2} who's members are individually referenced in the bridge-over-stream relation. This operation is implemented by generating code that filters the cross product of bridges and streams by removing the ones which exactly match tuples in the bridge-over-stream relation. This function is implemented within GINDB.

The new query forms for passing objects and regions into GINDB are as follows:
`(region` *region-constant-expr*`)`,
`(thing` *thing-constant-expr*`)`

where the constant expressions in the above are either

- an instance of a region or object
- an atom which is associated with a region or object in the GINDB database
- a string which is associated with a region or object in the GINDB database
- an arbitrary lisp form which produces a region or object

This flexibility was provided in order to maximally simplify use of GINDB as a function from lisp. In particular, it is now possible to type the following sequence of requests;

```
(setq foo (GINDB '(retrieve-region coniferous)))
(GINDB '(retrieve-region (OR (region ,foo) deciduous))
```

BRIDGES:

BR1
BR2
BR3
BR4

STREAMS:

ST1
ST2
ST3
ST4

BRIDGES-OVER-STREAMS:

BR1	ST2
BR4	ST3

HERE, THE NEGATION OF BRIDGES-OVER-STREAMS IS:

~BRIDGES-OVER-STREAMS:

BR2	ST1
BR2	ST4
BR3	ST1
BR3	ST4

Figure 5-13: A small relational data base

The net effect of which is to pass in the results of the previous query as an instance of a GINDB region. The syntax has proved quite helpful in use.

GINDB's internal database is essentially a trivial code construct for storing and retrieving regions and objects that are developed in a given GINDB session. This database is implemented as two association lists, where strings and atoms are allowed as binding labels. As we move to larger numbers of stored objects, more complicated database structures will be required.

There are a number of obvious additions to GINDB which can be incorporated in the future, but it is important to remember that our purpose is only to support the TAPS system as opposed to conduct R&D in the area of relational databases. As such, these new capabilities will be introduced on an as-needed basis. A brief list of the likely additions follows:

- the ability to pass back regions and objects from GINDB which are bounded in size
- a capability to store instances of relations, objects and regions on disk (such that they will not have to be regenerated on each session with GINDB).
- introduction of variables into the query language such that temporary results can be bound and accessed in separate portions of a large query form (the current syntax requires the user to repeat phrases)
- inclusion of various forms of meta-data into GINDB, such as
    - o introduction of forms attached to relations which compute instances of those relations
    - o the ability for the user to define new relations on the fly
    - o the ability to update relations, and have dependent relations flagged, or updated in turn

The first point in the above list admits to a simple correction. Currently, whenever GINDB returns a region it occupies the entire 512 x 512 field even if the feature is one element in size. This was done for simplicity in the first version of the

system, and can be corrected by returning bounding boxes for regions together with the appropriate registration information (to say where the bounding box lies). All the machinery is in place within GINDB to support this function, only a small effort will be required to make the change.

The capability to store instances of GINDB objects is clearly important in order to support a user-workstation model. At the current time, all relations, regions and objects which the user wishes to manipulate have to be computed each session at the Lisp machine, meaning that no discoveries can be regarded as permanent. The technical problem here is that it is not possible to store pointers on disk since the addresses they reference are tied to a given lisp image. We have solved this problem in other contexts by translated instances into a database-like record format. This will require a little attention to detail that we have preferred to bypass in the past.

It is important to note that even though we wish to make GINDB data persistent, we do not expect to expect to manipulate a tremendously large amount of data at any time. This translates into the claim/hope that GINDB can remain a core-resident database retrieval system, meaning that we can avoid the need to implement the caching and paging schemes (etc.) which go along with disk based database systems.

We have not examined the difficulty of introducing temporary variables into the query language beyond a cursory level of detail. It appears to be a change primarily for the code generation portion of GINDB, and there is some possibility that query variables would be difficult to include in the (deliberately simplistic) recursive-descent approach the code-generator currently applies. Since the intended purpose of the change is syntactic, the amount of energy we will devote to the problem is limited as well.

When TAPS is exercised in a workstation mode, it is reasonable to expect that its data will become more volatile, since new objects and relations may be produced on the fly. The ability to support these capabilities raises a collection of 'deep' relational database issues having to do with the inclusion of meta-data in a retrieval system.

In more detail, when a new object is created it is important (from the perspective of database completeness) to determine the relations the object ought to belong to. This in turn requires relations to know the domain of their attributes (so that the relevant relations can be identified), or schemas which link object types to the relations in which they participate. In addition, in order to automatically produce new instances of relations (new tuples given a single attribute), it will be necessary to attach methods to relations which perform the appropriate search. If these methods are viewed as constraint forms we gain the ability to update properties of objects and determine if they still belong to a given relation. This is important in support of the general goal of database consistency.

Without going into a long list of relational database issues and solution methods, it should be apparent that a variety of problems arise as the data becomes more volatile. From our perspective it is important to avoid such complications where possible, which suggests that GINDB should be used to represent fairly static facts about terrain features and relations between objects fixed in the terrain (objectives, areas of operation, minefields and other non-moving features).

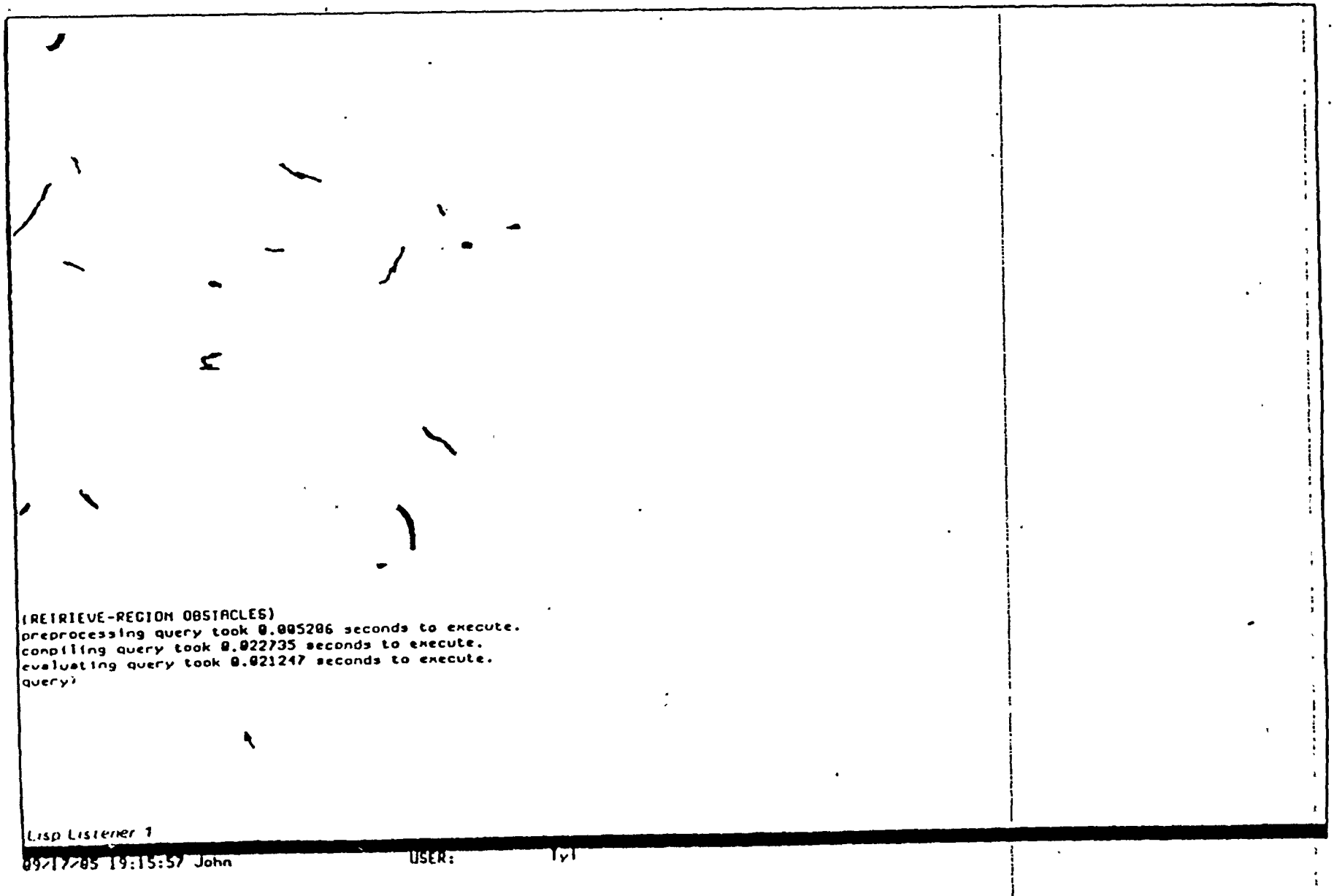
5.4.8 Example Queries

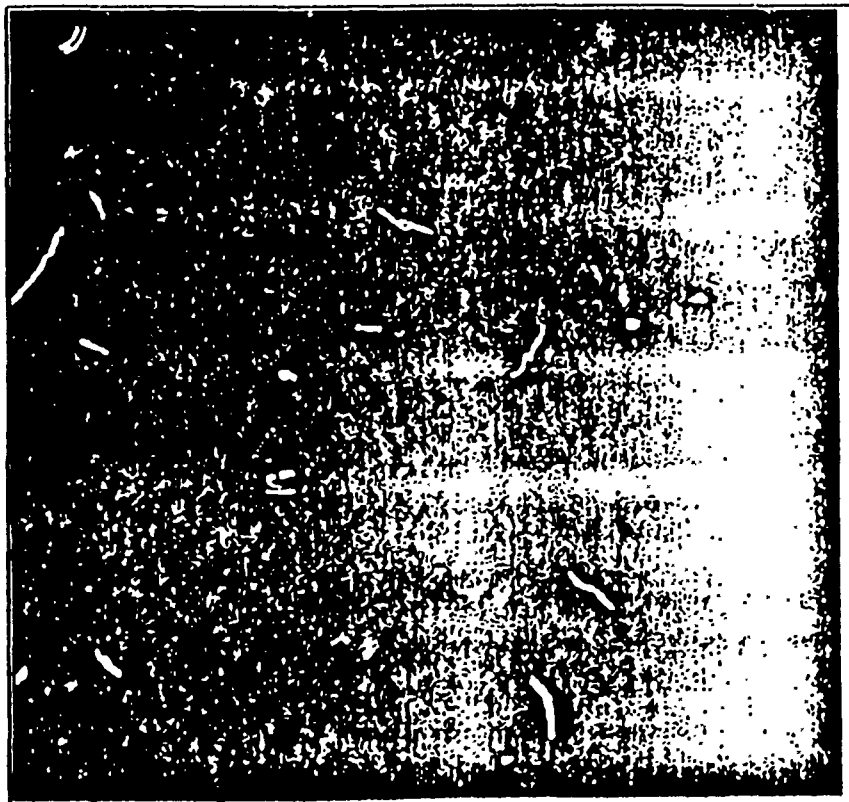
The following pages present output from the GINDB system, generated as images of the Symbolics display. The use of GINDB is also documented in the context of our demonstration of TAPS (see appendix III), but these examples (though with toy data) focus more clearly on GINDB. Note that each example provides the input query, graphical and/or textual output, and timing figures for each of the query preprocessing, compilation, and execution stages.

The sample database contains a number of regions culled from CATTS data, and a library of approximately 200 objects which were created solely to exercise the query language and have no military meaning. There are four object types in this data; letters, hills, obstacles, and unknown-objects. The first three are objects with spatial components taken from character fonts. The last comes from a mobile robot example,

where occlusions during sensor sweeps result in conical areas that may include unknown obstacles.

In the material that follows, the command *show* was used to display the spatial components of the objects returned by the previous query.

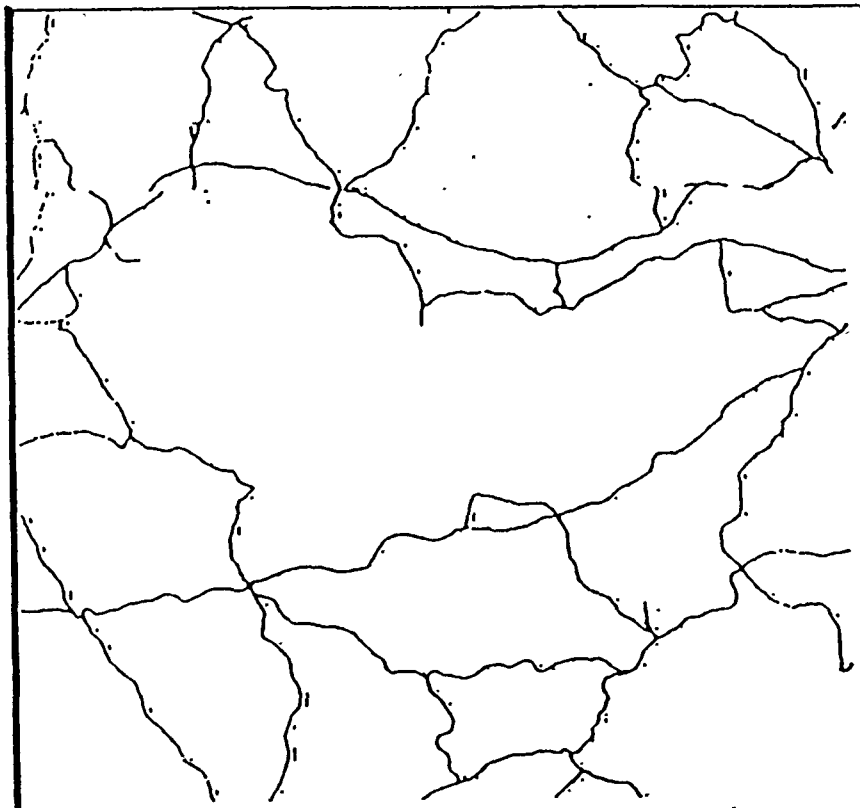

(RETRIEVE-REGION OBSTACLES)
preprocessing query took 0.005206 seconds to execute.
compiling query took 0.022735 seconds to execute.
evaluating query took 0.021247 seconds to execute.
query>
Lisp Listener 1
09/17/05 19:15:57 John
USER:



```
(RETRIEVE-REGION (NOT OBSTACLES))
preprocessing query took 0.005493 seconds to execute.
compiling query took 0.026105 seconds to execute.
evaluating query took 0.016027 seconds to execute.
query> 
```

Lisp Listener 1

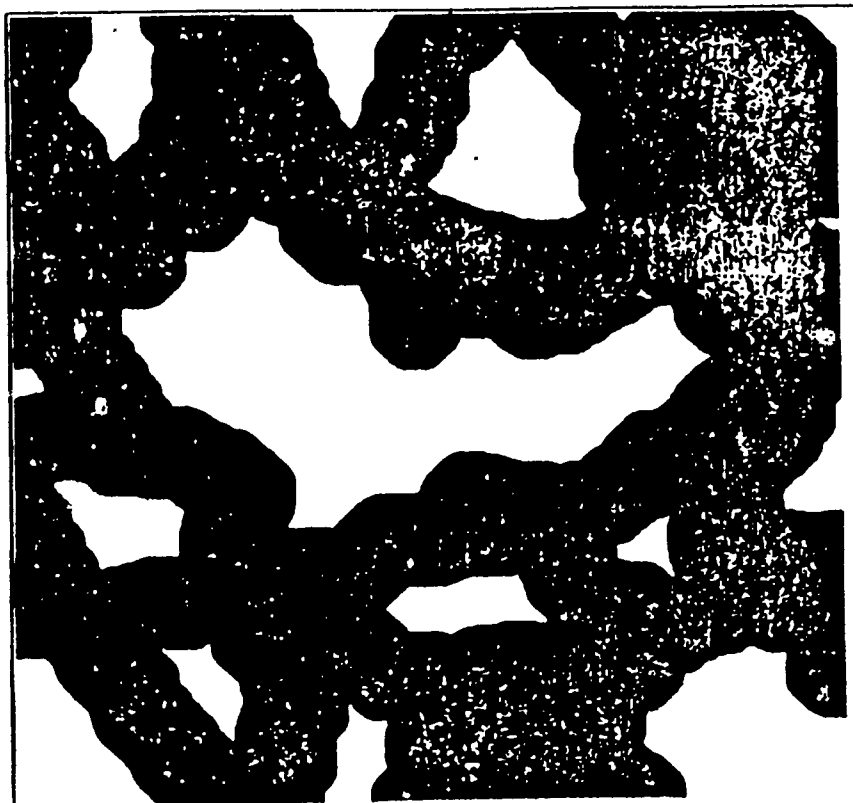
09/17/85 19:17:02 John USER: Tyl



```
(RETRIEVE-REGION SECONDARY-ROADS)
preprocessing query took 0.005 seconds to execute.
compiling query took 0.022506 seconds to execute.
evaluating query took 0.017559 seconds to execute.
query> 
```

Lisp Listener 1

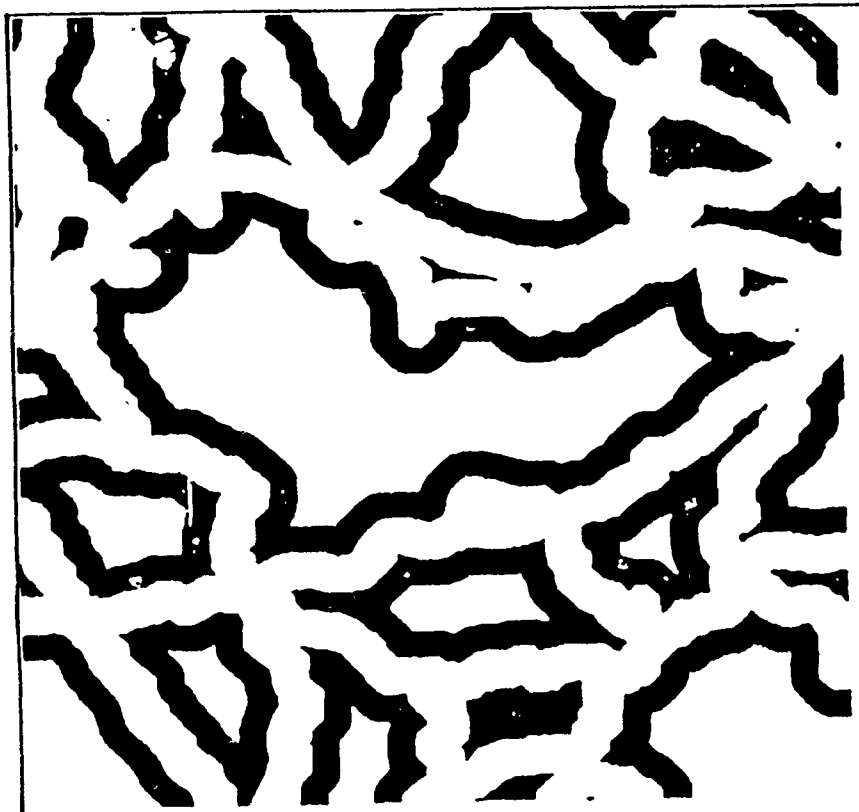
09/17/85 19:18:01 John USER: Yyl



```
(RETRIEVE-REGION (NEARBY SECONDARY-ROADS 20))
preprocessing query took 0.005913 seconds to execute.
compiling query took 0.04044 seconds to execute.
evaluating query took 4.64899 seconds to execute.
query>
```

Lisp Listener 1

09/17/85 19:19:01 John USER: Tyi



```
(RETRIEVE-REGION (XOR (NEARBY SECONDARY-ROADS 8) (NEARBY SECONDARY-ROADS 20)))
preprocessing query took 0.009506 seconds to execute.
compiling query took 0.103268 seconds to execute.
evaluating query took 6.86968 seconds to execute.
query>
```

Lisp Listener 1

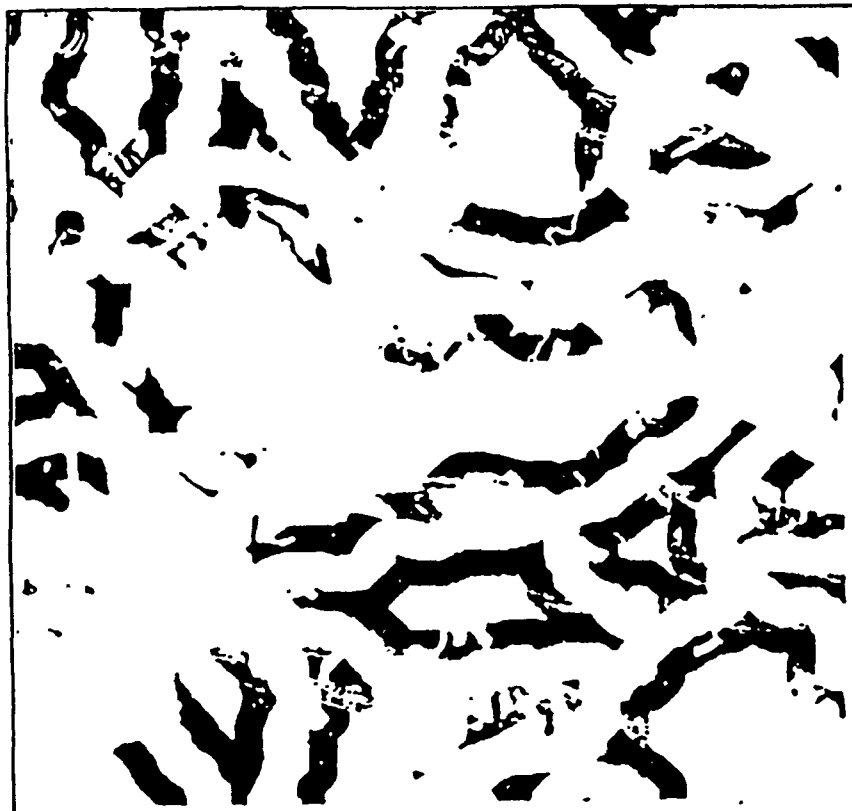
09/17/85 19:25:14 John USER: tyi

```
(RETRIEVE-REGION (SLOPE 0 2))
preprocessing query took 0.004713 seconds to execute.
compiling query took 0.029168 seconds to execute.
evaluating query took 3.72674 seconds to execute.
query> 
```

Lisp Listener 1

09/17/85 19:27:21 John USER: Tyi

The areas returned by this query are potential locations for regimental headquarters. They have moderate slope, are between 200 and ~400 meters from secondary roads and are on traversable terrain which excludes known obstacles.

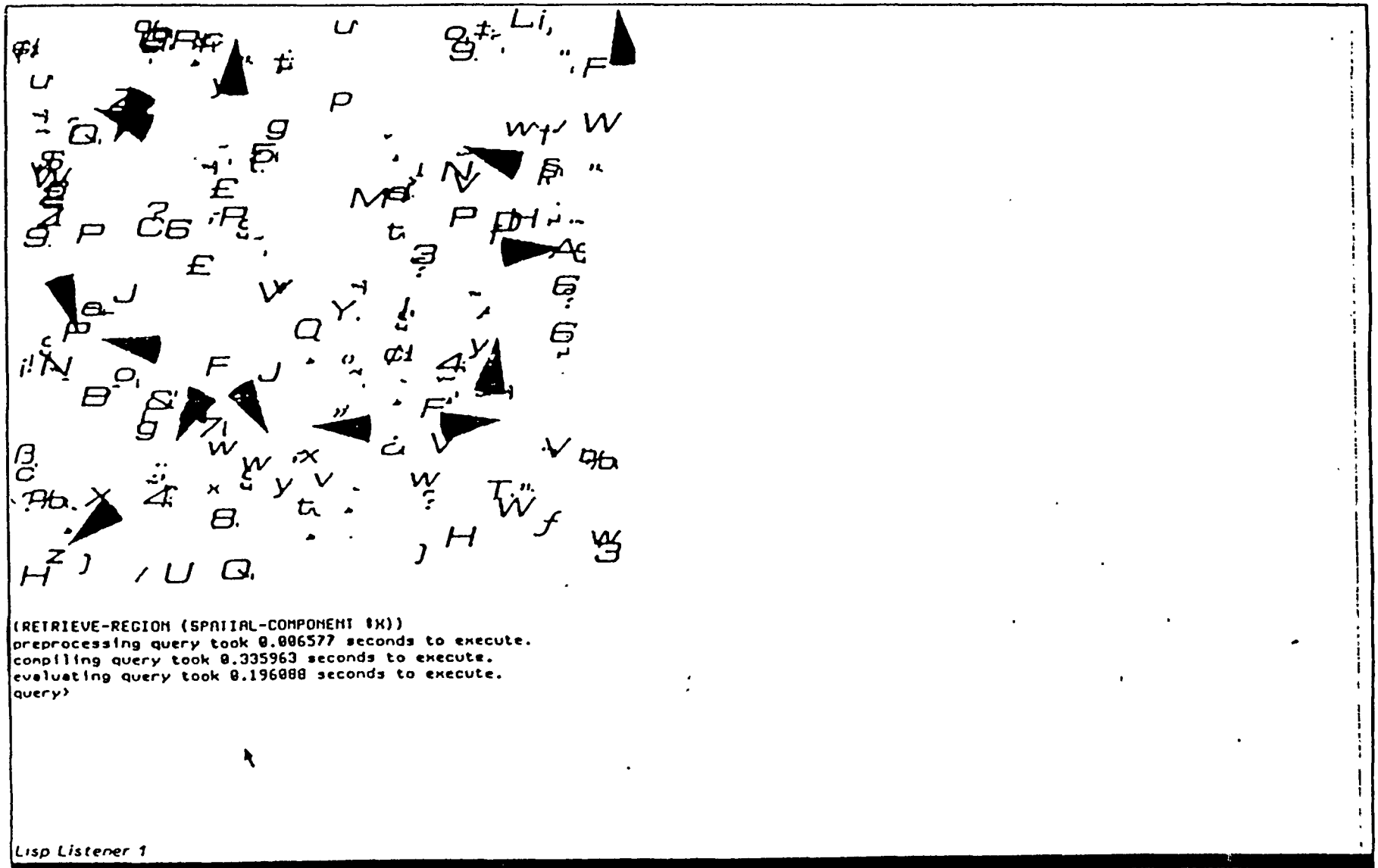


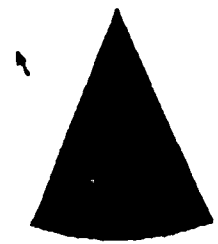
```
(RETRIEVE-REGION (AND (NOT OBSTACLES)
                      (NOT URBAN-AREAS)
                      (NOT TRAV3)
                      (XOR (NEARBY SECONDARY-ROADS 0) (NEARBY SECONDARY-ROADS 20))
                      (SLOPE 0 6)))
preprocessing query took 0.023793 seconds to execute.
compiling query took 0.312453 seconds to execute.
evaluating query took 10.9559 seconds to execute.
query> ■
```

Lisp Listener 1

09/17/85 19:25:25 John USER: Tyl

This is the library of known objects.





```
(RETRIEVE-REGION (IM-DIRECTION 250 250 -90 45 150))
preprocessing query took 0.006445 seconds to execute.
compiling query took 0.027898 seconds to execute.
evaluating query took 0.065849 seconds to execute.
query>
```

Lisp Listener 1

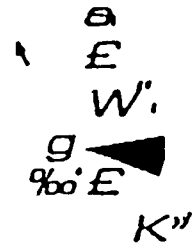
10/01/05 13:43:49 Dan USER: (y)

This finds the objects which overlap the triangular template.

```
10 things retrieved for query.
(RETRIEVE-THING $X (IN-REGION $X (IN-DIRECTION 250 250 -90 45
150)))
preprocessing query took 0.011332 seconds to execute.
compiling query took 0.823857 seconds to execute.
evaluating query took 0.974547 seconds to execute.
query
```

Lisp Listener 1

6/17/85 20:07:22 John USER: Tyl

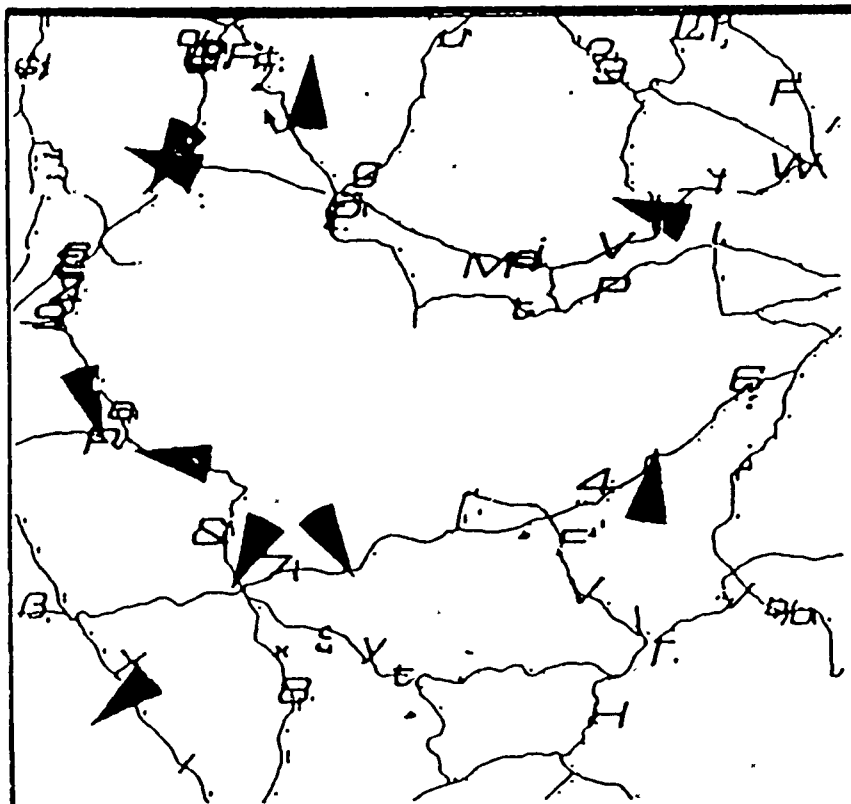


(LIST OF RESULTS)
query>

Lisp Listener 1

10/01/85 13:42:53 Dan USER: [y]

This query finds all objects whose spatial part overlaps secondary roads, and displays them on top of those roads.

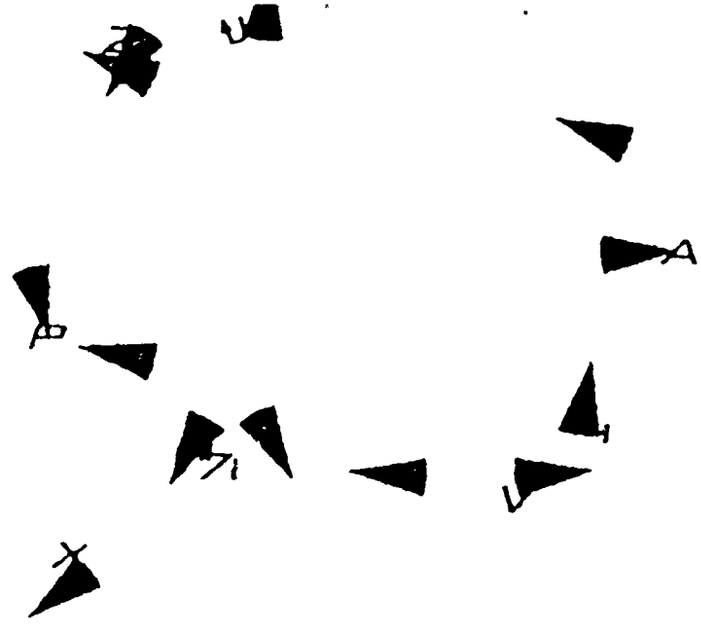


```
(RETRIEVE-REGION (XOR (SPATIAL-COMPONENT $X) SECONDARY-R
OADS) (IN-REGION $X SECONDARY-ROADS))
preprocessing query took 0.02288 seconds to execute.
compiling query took 1.90497 seconds to execute.
evaluating query took 1.11804 seconds to execute.
**MORE**

Lisp Listener 1
09/17/85 20:09:25 John          USER:          Tyi
```

This query displays all objects which are conceptually in regions (objects) of type = "unknown". This includes the objects of type = "unknown" themselves.

```
evaluating query took 1.0082 seconds to execute.
query
```



```
(RETRIEVE-REGION (SPATIAL-COMPONENT $X)
                 (AND (IN-REGION $X (SPATIAL-COMPONENT $Y))
                      (UNKNOWN $Y)))
preprocessing query took 0.020942 seconds to execute.
compiling query took 0.877387 seconds to execute.

Lisp Listener 1
09/17/85 20:20:15 John          USER:          Yp1
```

5.5 The World Model

The material in this section discusses the information we expect a terrain reasoning system to maintain about the military environment; the organization and composition of units, the capabilities of vehicles and individual weapons systems, as well as the capabilities of units taken as a whole. In addition, we postulate a collection of feature models which express the characteristics that must be possessed by the terrain intended to fulfill particular tactical purposes. The motivation for incorporating this knowledge into TAPS was discussed in chapter 3. In brief, we view the resulting database as the minimal set of support for a wide body of applications.

The discussion below documents design ideas as opposed to implemented code.

5.5.1 Representation of Force Units and Equipment

In order to resolve questions concerning the use of terrain, it is important to model the capabilities of the units, vehicles, and weapons systems which are deployed on the terrain. The most obvious form of representation is a system of frames, which are flexible templates for representing typical situations. Frame languages are a well explored technology for manipulating class structured data, and are therefore particularly well suited to the hierarchical structures found in military organizations. Their ability to support deduction by inheritance provides a desirable parsimony in representation.

It's quite easy and natural to define units and equipment in terms of frames. Consider, for example, the following simple frame for a mechanized rifle squad,

```
frame: Mech Inf Rifle Squad
   slot: amo     value: Mech Inf Plt
   slot: members   if-needed: AskFor( Mech Inf Squad Leader )
                              AskFor( Assistant Squad Leader )
                              AskFor( Mech Inf Gunner )
                              AskFor( Mech Inf Driver )
                              AskFor( Antiarmor Specialist )
                              AskFor( Automatic Rifleman , A )
                              AskFor( Automatic Rifleman , B )
                              AskFor( Grenadier )
                              AskFor( Rifleman/Sniper )
    slot: vehicles  value: BradleyFightingVehicle
```

where "amo" means "a member of", and "if-needed" refers to a program that can be invoked to determine the value of an individual's frame slot, if that value is not present with the individual and is needed to answer a query, or perform a computation, etc. "AskFor" refers to a program that queries the user for the value of a slot; it will be invoked, typically, when a new frame is created for a new individual (i.e. instance) of a class. We use a loose pseudo-code here for the sake of readability. Typically, frames are specified as embedded association lists in lisp.

The utility of this form of representation can be seen from figure 5-14 (where "amo" means "a member of", "aio" means "an individual of", and "ako" means "a kind of"). Using this structure, it is possible to deduce properties of units which are not expressed local to their definition. So, for example, if we want to know the area controlled by a mechanized rifle platoon, we can look down the AMO links to discover it is composed of squads employing Bradley fighting vehicles, and up the AIO links from there to the definition of a Bradley vehicle which will identify the range of its weapons systems.

Notice also that a complete frame database for battalion organization and equipment forms a "tangled" hierarchy, with a great many cross linkages (of different link types) between frames. Mechanisms for performing inheritance through tangled hierarchies are common in object oriented programming langauges.

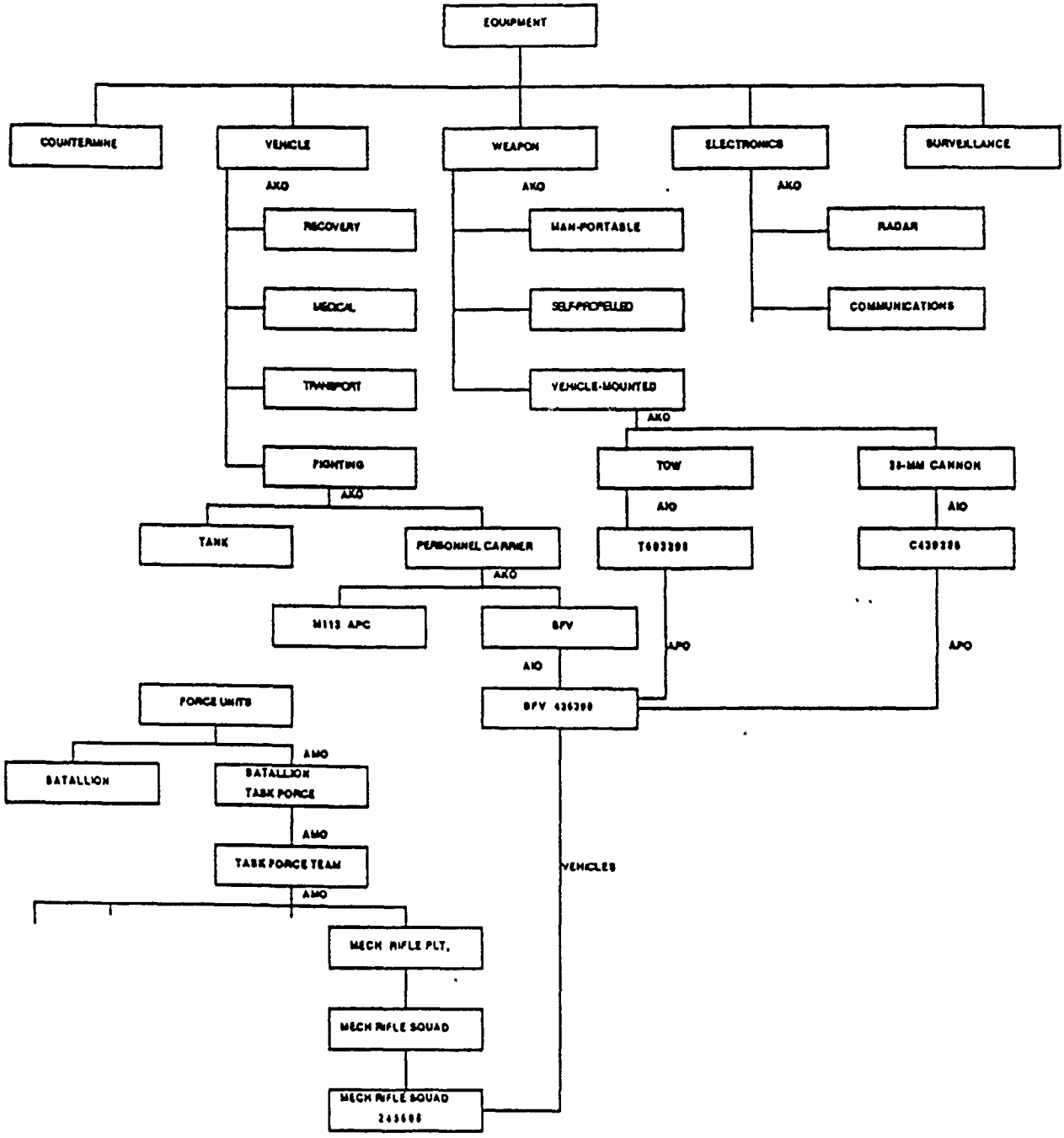


Figure 5-14: A Portion of a Network Representing Battalion Equipment

5.5.2 Tactical Feature Models

As mentioned in the introduction, the goal of including tactical feature models within TAPS is to raise the level of discourse available for conducting terrain analysis. In specific, the object is to provide a vocabulary for terrain features of military interest and allow the user to form queries using that terminology as opposed to the very detailed constructions that GINDB supports.

One difficulty with constructing feature models is that they seem to be heavily context dependent; even something so basic as the concept of navigable terrain differs according to the entity being moved (type of vehicle or personnel) and according to echelon, where corridor width and terrain fatigue considerations come into play. Terrain features such as suitable defensive areas have similar sources of variation, and because of the diversity of tactical situations they are likely to be much more difficult to define.

A key observation about these abstract features comes from watching analysts perform their jobs; as we mentioned in section 5.1 they are typically asked to assess the impact of terrain on specific tactical actions, and given that information as input they know the set of features that are required/desired. Said in a different way, terrain features of military interest can be classified by *tactical operation* and *terrain role*. An example (which we develop further below) is the river crossing operation also referenced in section 5.1. Here, the operation is a crossing, and the terrain roles are *crossing point*, *covering force site*, *potential enemy force site*, *ingress path*, *egress path*, and perhaps a *beach-head site*, meaning a defensive site to occupy on reaching the opposite side of the river.

In performing terrain analyses for operations of this kind, the analyst will also know the nature (or specific type) of the vehicles, and friendly units involved. So, for a river crossing he may be directed to find sites where no bridging equipment is required, or he may be allowed to utilize knowledge of a particular mobile bridge. It may be that the analyst also employs knowledge of enemy units in the area such as their

approximate location and nature (we haven't determined if this is a standard input of terrain analysis). In any case, this information will have to be transported to the tactical feature models in some form; obvious approaches are to parameterize the models to account for the current context, to assume a particular resource pool as a default, or to separate the models into subclasses where specific equipment and resources are assumed.

To complete the analogy, given the tactical operation, knowledge of the current context, and a set of terrain roles to fill, the analyst's next task is to identify terrain regions which have the desired characteristics. In our approach, this process will be accomplished by employing GINDB queries which in turn access the multi-attribute feature data we are assuming available for the local terrain. (In general, the analyst has less information to work with, and has to spend a significant amount of time inferring information of the kind CATTS supplies). These queries will result in candidate regions which the user can accept, prune or reject as desired. The system might also return more than one candidate for a given terrain role, meaning that the user might wish to apply additional criteria (via GINDB) in order to narrow the solution-set further.

By embedding GINDB queries in parameterized models of this kind, the effect is to automatically translate expressions for abstract features of military interest into the right set of low level queries concerning the terrain. This approach should be of significant benefit because:

- it allows the user to interact with the terrain analysis system in a vocabulary that is natural, and at the right level of abstraction for the task
- it shields the user from the details of the underlying query language and database concerning the terrain
- it provides a formalism for defining a wide variety of features, and for sharing definitions of component features among many models

We have examined some alternatives for the design of these tactical feature models. We discuss our current understanding of their structure below.

5.5.2.1 A Sample Tactical Feature Model

From the above discussion it is clear that tactical feature models must contain at least the following four parts:

- the tactical operation of interest
- the terrain roles required by the operation
- a set of queries which identify regions that can fulfill those roles
- a method of specializing the queries in the model to account for the military context (e.g., units and equipment) involved

A deliberately simplified instance of such a model is shown in figure 5-15. This figure presents a portion of the river crossing site model mentioned above using a hypothetical syntax for attaching queries to roles in models. The overall form is patterned after procedures in a frame language which accept parameters and are invoked by if-needed forms. We have introduced a new GINDB form, *make-thing*, which creates an instance of an object given a terrain region (in the current implementation, objects can only be constructed external to GINDB and then retrieved). We have also allowed ourselves to embed GINDB requests in logical conditionals, but we have otherwise stayed within the language capabilities described so far.

The interesting thing about this example is that it points out both the merits of the feature model approach and a number of problems which still need to be solved. At this stage we have explored some alternative model designs (discussed below), but have not implemented any of the potential solutions.

On the side of merits, it is clear that a simple procedural/frame format can capture the essence of something as complicated as this river crossing example, and is easily parameterized to account for differing military context. In addition, the simplicity of the example (in structure, not syntax) supports the paradigm of identifying complex features via low level manipulations of terrain properties using a language such as GINDB.

Figure 5-15: A tactical feature model for a river crossing site

```
Model:  River-crossing-site (river equipment-to-move
                             bridging-equipment from-side to-side)

   Role: crossing-point
   Query: If equipment-to-move = 'men
          then
             ;the site should be flat and very shallow
             (make-thing crossing-point
                         (and (spatial-component river)
                              (depth 0 4)
                              (slope 0 2)))
          else if equipment-to-move = 'tanks
               then  ;no bridges then tolerate 4 degrees and 6 feet
                     ;Examine river width and bridge type if mobile
                     ;bridges are available.
               ...

   Role: enemy-force-site
   Query:
          ;must be near crossing point on the far side of the river
          ;and have field of fire over it
          (make-thing enemy-force-site
             (and (nearby crossing-point 20)
                  (spatial-component h)
                  to-side
                  forested)
             (and (field-of-fire h crossing-point)
                  (hill h)))

   Role: covering-force-site
   Query: ;same as above except it must be on the from-side of the
          ;river and have field-of-fire over the enemy-force-site
          (make-thing enemy-force-site
             (and (nearby crossing-point 20)
                  (spatial-component h)
                  from-side
                  forested)
             (and (field-of-fire h enemy-force-site)
                  (hill h)))
```

However, there are additional requirements on the models in the context of TAPS. In particular, since TAPS is an interactive system it is reasonable to expect that the user will supply partial information about the units and equipment involved (own and enemy) in the tactical operation. Rather than produce no output at all, the desired result would be for TAPS to return a region that satisfies the constraints which are currently known. I.e., given partial input, TAPS should produce partial output. In this approach, we can view the interactive process as a sequential refinement exercise, moving from rough sketches of the feature of interest to the best instantiations possible.

Next, since TAPS is also a mixed initiative system, it is perfectly reasonable for the user to supply important data in different orders. This implies that the model instantiation code must be capable of recognizing and running those queries whose support data is available. This argues for an expression of tactical feature models built along the lines of a constraint processing language (in the manner of the KNOBS or TEMPLAR systems [Millen 83, Shapiro 84]) which distinguishes the following concepts;

- constraint readiness (when all the parameters of a constraint are known)
- constraint evaluation (to determine if a supplied region is acceptable)
- constraint inversion (i.e., to build regions which satisfy constraints; this is what GINDB queries currently supply).

In this approach, we reinterpret the query forms attached to terrain roles as constraints on the regions that can be slot fillers. We then implement an interpreter which is tasked to watch the interaction with the user and run evaluative constraints when the data is supplied, or run those generative versions which are ready when role instantiation is requested.

The general form for feature models built in this image is shown in figure 5-16. Note that this construction (with attached interpreter) addresses the mixed initiative issues mentioned above, but also raises new questions concerning model application. For example, in the event that more than one method (for object generation or instantiation) becomes applicable at the same time, which one should be run? If all are

Figure 5-16: A Constraint Based View of Tactical Feature Models

```
Model:  model-name(params)

   Role: role-name
   Role Acceptance Criteria:
      constraint1
      constraint2
      ...

   Instantiation Methods:
      method-name1: query
      method-name2: query2
```

applied, does the system carry the multiple answers as alternative slot fillers, or is a method introduced for extracting only the *best*? (This situation can arise even in the first feature model example). Given the interactive refinement usage postulated, it seems reasonable for the system to maintain alternative instantiations which the user can then restrict. It would also make sense to give the system some sense of the *most informed* method, which presumably depends upon the most specific parameter data available.

5.5.2.2 Evaluating and ranking role fillers

Another important form of interaction alluded to in the above occurs when the user independently creates a region and asserts it to be an instantiation of some terrain role. In this situation, it would be very desirable for TAPS to evaluate the user's choice based on the knowledge contained in its feature models.

There are a number of ways to provide this capability. On the simple end, we can define the ranking of a user-supplied region as the difference between it and the system's own notion of what ought to be present. Since GINDB is well set to manipulate individual pixels, we can cast this operation in terms of a pixel count. That is, we run the query attached to the terrain role on the region the user supplies, and then calculate the percentage of pixels both the model and the user find acceptable. The percentage of agreement measures the goodness of the user's proposal.

This type of analysis can be expanded to factor in both agreement and disagreement. So, for example, when the user proposes a role filler, TAPS could run the model based query on an area surrounding the user's input, and then weight the four possible pairs of outcomes differently (e.g., where TAPS accepts a pixel the user considers invalid, or where TAPS doesn't retrieve one the user does, etc.). The argument with this type of approach is that it is reductionist, and adds little insight into the reasons which determine the relative value of a potential slot filler.

There are a range of approaches to candidate evaluation which provide a better sense of well-informed judgement. The simplest among them is to build a calculus for ranking terrain feature instantiations based on weighted production rules. We could then write deductive expressions of the form:

```
(implies crossing-point (and (spatial-component river)
                             (depth 0 4)
                             (slope 0 2))
                        .8)

(implies crossing-point (and (spatial-component river)
                             (depth 0 4)
                             (slope 0 1))
                        .9)
```

and apply them to the region supplied by the user. The result is a ranking of candidate regions which admittedly depends upon magic numbers, but at least those numbers were obtained from the user.

A more satisfying solution is to employ a form of the type shown in figure 5-17. Here, the user supplies new information which explicitly defines an ordering of acceptance criteria, i.e., that low exposure to enemy fire is more important than the depth of the stream at the crossing point when multiple candidates are supplied. This approach is equivalent to the mechanism employed by TEMPLAR ([Shapiro 84]), which partitions constraint forms into categories of *hard* and *soft*. Here, *hard* constraints are mandatory requirements on any slot filler, while *soft* constraints are optional. The

Figure 5-17: Ranking Criteria for Feature Instantiations

```
Role: Crossing Point
   Method: when-men-must-wade
   Query: (and (spatial-component river)
               (depth 0 4)
               (slope 0 2))

   Ranking Criteria:
      first: exposure to enemy fire is low
      second: length of crossing is short
      third: depth at crossing point is less than 3 feet
```

candidate ultimately selected must pass all of the first type, and as many of the soft constraints (in ranked order) as possible.

A final note is that this constraint oriented approach requires a considerable quantity of new mechanism and should therefore be examined carefully. We plan to explore it, along with other languages for expressing feature models in future extensions to the BCA research.

6. Future Directions

There are two obvious directions for continuing the work described in this report. The first is a research oriented extension aimed at completing the design and implementation of tactical feature models, and the second is concerned with adding breadth to the TAPS system, and bringing it into user communities within the military. These ideas are mutually supportive, and are described in more detail below.

6.1 Tactical feature model design and development

The creation of tactical features models is the next critical step in the development of TAPS. These models will provide a language for describing terrain features of military interest (beyond the powerful, but admittedly cumbersome vocabulary of GINDB), and form the basis for an interactive mechanism for instantiating features in the context of a terrain analysis/planning session. Our current ideas on the structure of tactical feature models were presented in chapter 5. The principal tasks associated with this line of effort are listed below:

- Develop a computation model for representing tactical feature models with the following properties: they input parameters describing the current order of battle, they support partial descriptions of desired features, and they are organized around the different functional roles for terrain in specific military operations.

- Construct a mixed initiative method for instantiating feature models that allows the user to express queries in the functional vocabulary described above, as opposed to the GINDB syntax which is currently employed. (The models will employ GINDB queries internally.)

- Build a sample library of such models, and demonstrate their use.

The benefit of this work will be a demonstrable capability to support terrain analysts via artificial intelligence techniques, and a deeper understanding of the role terrain reasoning can play in supporting battalion level decision functions. As examples, the following application tasks could be solved through use of tactical feature models in an expanded version of TAPS:

- Identify the terrain sites important for a river crossing operation, given the specific equipment and tactical scenario involved. This requires instantiating component sites, including potential enemy positions, exact locations for fording or bridging the river, and own force sites for securing an approach to the river.

- Propose placement of minefields. This requires identification of choke points within enemy avenues of approach.

- Develop an avenue of approach towards a specific objective. Key terrain, laterally inhibiting terrain, obstacles and categories of mobility zones (by vehicle type) are all component parts of avenues of approach. These can be identified by queries attached to functional roles within an avenue of approach model.

Several additional tools will have to be implemented in order to support applications of this kind. In particular, we will need to construct a representation for military organization and equipment which captures their capabilities, write routines for extracting features from elevation data (direct and indirect fields of fire, ridge lines, etc.), and develop an order of battle display. It is worth pointing out that a wide variety of image processing techniques can be applied to manipulating elevation data.

6.2 Applications level extensions of TAPS

Based on comments we have received after demonstrating TAPS, there is a role for pushing the current capabilities further toward practical applications. With attention to system level issues, it would be possible to take the current display and retrieval capabilities, increase their generality, and host them on processors designed to increase their speed and data handling capabilities. The result would be an applications prototype that could be employed by trained members of the terrain analysis staff. We believe such a system would be applicable at echelon from battalion through corps, and that it would be powerful enough to significantly impact current military processes.

To transit the TAPS system into application, several design changes are indicated. First, the retrieval language needs to be standardized and substantially clarified, to make terrain manipulation queries more accessible to less-than-expert users. A

possibility is to modify an existing relational language to incorporate logical operations on terrain. Second, the host for symbolic processing (the application system) should be separated from the host(s) which supports the terrain data and query language. By introducing this change, we identify a separable *terrain server*, which opens up the possibility for optimization as well as standardization of terrain processing requests. Finally, since the Multiple Pane Interface has already been implemented as a stand-alone component, it would make sense to explore options for optimizing its performance via different display and computation hardware. It operates quite efficiently at the current time, although we are clearly pushing the limits of the Symbolics system's display capabilities.

This applications effort would clearly benefit from a parallel research thrust, which would provide new capabilities that could be transitioned to the development prototype as they became available. A benefit in the reverse direction is that early prototypes of an applied system would provide an effective host for further research activities.

References

[Havens 83] Havens, William & Alan Mackworth.
Representing Knowledge of the Visual World.
Computer 16(10):90-96, October, 1983.

[Kuan 84] Kuan, Darwin T., Rodney A. Brooks, James C. Zamiska, and Mangobinda Das.
Automatic Path Planning for a Mobile Robot Using a Mixed Representation of Free Space.
In *Proceedings, First Conf. on AI Applications*, pages 578-584. December, 1984.

[Loberg 86a] Loberg, G.
Preliminary Report on a Knowledge Engineering Experiment.
Technical Report IR-0001, Advanced Information Processing Division, COMM/ADP, US Army Communications-Electronics Command, August, 1986.

[Loberg 86b] Loberg, G.
Acquiring Expertise in Operational Planning: A Beginning.
Technical Report IR-0002, Advanced Information Processing Division, COMM/ADP, US Army Communications-Electronics Command, August, 1986.

[Millen 83] Millen J. K., Engelman C., Scarl E. A., Pazzani M. J.
KNOBS Architecture (Draft).
Technical Report, The MITRE Corporation, 1983.

[Payne 86] Payne, J.R., Stachnick, G.L., Rosenschein, and Shapiro, D.
Operations Monitoring Assistant System Design.
Technical Report TR-1113-1, Advanced Decision Systems, July, 1986.

[Quattromani 82] Quattromani, Anthony F.
Catalog of Wargaming and Military Simulation Models.
Technical Report, Studies, Analysis, and Gaming Agency Organization of the Joint Chiefs of Staff, May, 1982.

[Samet 83] Samet, Hanan.
The Quadtree and Related Hierarchical Data Structures.
Technical Report TR-1329, University of Maryland dept. of Computer Science, November, 1983.

[Samet 84] Samet, H., A. Rosenfeld, C. Shaffer, R. Nelson, Y. Huang.
Application of Hierarchical Data Structures to Geographic Information Systems: Phase III.
Technical Report TR-1457, University of Maryland dept. of Computer Science, November, 1984.

[Shapiro 84] Shapiro, D.G., McCune, B.P, Courand, G.J., Wilber, B.M., Payne, J.R, Kefalas, J., Hale, C.R., Finger, J., and Wilensky, R.
TEMPLAR (Tactical Expert Mission Planner) Design.
Technical Report TR-84-134, Rome Air Development Center, June, 1984.

[Shapiro 85a] Shapiro, Daniel G., Philip S. Marks, Jeffrey M. Abram.
Battlefield Commander's Assistant.
Technical Report TM-1058-01, Advanced Information & Decision Systems, 201 San Antonio Circle, Suite 286, Mountain View, California 94040-1270, May, 1985.

[Shapiro 85b] Shapiro, Daniel G., Woodfill, J.
GINDB, A Geographically Intelligent Database.
Technical Report TM-1058-02, Advanced Decision Systems, 201 San Antonio Circle, Suite 286, Mountain View, California 94040-1270, October, 1985.

[Shapiro 86] Shapiro, Daniel G., Tollander, Carl.
The Design of TAPS.
Technical Report TM-1058-03, Advanced Decision Systems, 201 San Antonio Circle, Suite 286, Mountain View, California 94040-1270, August, 1986.

[Stachnick 87] Stachnick, G.L., Applebaum, L.A., Marks, P., Marsh, J., Rosenschein, J., Schoppers, M., and Shapiro, D.
Airland Battle Management Planning Study.
Technical Report TR-1127-01, Advanced Decision Systems, January, 1987.

[Ullman 82] Ullman, Jeffrey D.
Principles of Database Systems.
Computer Science Press, Rockville, Maryland, 1982.

[Weiler 80] Weiler, Kevin.
Polygon Comparison Using a Graph Representation.
Sigraph , 1980.

I. BCA Bibliography

In the process of conducting our domain survey, we collected (and examined) the following documents on military theory, battalion (and other echelon) operations, and existing application systems.

- Airland Battle 2000, Army Training and Doctrine Command, Ft Monroe, VA 23651, Aug 1982

- Bentley, Jon L., "Multidimensional Binary Search Trees used for Associative Searching", *Comm. of ACM*, Vol. 18, No. 9, September 1975, pp. 509-517.

- Bentley, Jon L., "Multidimensional Binary Search Trees in Database Applications", *IEEE Trans. on Software Engineering*, Vol. SE-5, No. 4, July 1979, pp. 333-340.

- Blumenthal, D. K. and others, "An analysis of a Massed Versus a Dispersed Attacking Ground Force in Close Combat", Lawrence Livermore National Laboratory, 1984

- Campen, Timothy, "Application of Artificial Intelligence to Tactical Operations", HQ, Joint Special Operations Command

- "Conference on Command & Control Decision Aids, Vols I-VII", Rome Air Development Center, Griffiss AFB, New York, Nov 1983

- Cooper, Dennis and others, "C^2 Enhancement of CORDIVEM", General Research Corp., P O Box 6770, Santa Barbara, CA 93111, July 1983

- Lybrand, J. P., "An Analysis of the Use of Decoys in Defense Against a Massed Attack", Lawrence Livermore National Laboratory, October 1984

- FM 7-10 (Infantry Rifle Company), Headquarters, Dept of the Army, Washington, DC

- FM 7-20 (The Infantry Battalion), Headquarters, Dept of the Army, Washington, DC

- FM 30-5 (Combat Intelligence), Headquarters, Dept of the Army, Washington, DC

- FM 71-2 (The Tank and Mechanized Infantry Battalion Task Force), Headquarters, Dept of the Army, Washington, DC

- Havens, William & Alan Mackworth, "Representing Knowledge of the Visual World", *Computer*, Vol. 16, No. 10, October 1983, pp. 90-96.

- Intelligence Preparation of the Battlefield (TC34-3), Army Intelligence Center and School, Ft. Hauchuca, AZ 85613

- Kuan, Darwin T., Rodney A. Brooks, James C. Zamiska, and Mangobinda Das "Automatic Path Planning for a Mobile Robot Using a Mixed Representation of Free Space", in *Proc. of the First Conf. on AI Applications*, December 1984, pp. 578-584.

- McKeown Jr, David., "MAPS: The Organization of a Spatial Database System Using Imagery, Terrain, and Map Data", Carnegie-Mellon University Report CMU-C5 - 83-136, July 1983.

- Quattromani, Anthony F., "Catalog of Wargaming and Military Simulation Models", Studies, Analysis, and Gaming Agency Organization of the Joint Chiefs of Staff, May 1982.

- Rebane, G. J. and others, "Dynamic Displays for Tactical Planning, Vol I", Army Research Institute for the Behavioral and Social Sciences, April 1980

- RB 101-5, U.S. Army Command and General Staff College, Fort Leavenworth, Kansas, May 1983.

- Samet, Hanan, "The Quadtree and Related Hierarchical Data Structures", University of Maryland Computer Science TR-1329, November 1983.

- Samet, H., A. Rosenfeld, C. Shaffer, R. Nelson, Y. Huang "Application of Hierarchical Data Structures to Geographic Information Systems: Phase III", U. of Md. Comp. Sci. TR-1457, November 1984.

- Thompson, George and R. Weeks, "A Common Methodology for CORDIVEM", General Research, P O Box 6770, Santa Barbara, CA 93111, July 1983

- Weiler, Kevin, "Polygon Comparison Using a Graph Representation", *Sigraph '80*, CMU.

II. SMAP Output

The slides attached to this appendix show the type of output it was possible to produce with the SMAP system, described in section 4-9. A discussion of each slide is provided below. Note that only three copies of this report contain a set of these color 35 millimeter slides.

The SMAP system provided a mechanism for displaying desired subsets of CATTS data (feature and elevation both) in user-selected combinations. In recognition of the fact that some types of data are space filling, while other CATTS fields encode predominantly linear features, the display menus are segregated into *background* and *overlay* options. SMAP could then display any number of overlays above any single background feature.

Slide 1: shows the groundcover background. A key describing the color coding of each groundcover subtype appears beneath the image.

Slide 2: This slide shows roads, river, and rail lines overlayed on the groundcover background from the previous slide. Documentation for the subtypes of each linear feature automatically appear.

Slide 3: Countour lines computed from the elevation data are added on top of the material in slide 2. This slide illustrates the visual clutter which results from the attempt to display too much information simultaneously. The need for flexibility in generating such displays was a major motivation behind SMAP.

Slide 4: This slide shows the CATTS "obstacle" field overlayed on a different background, which encodes a version of cross country mobility.

Slide 5: This slide shows river data overlayed on the elevation map, where the shading from green to red encodes increasing elevation. Note that this image gives a good feel for the landforms which was not quite present in previous pictures. It is clear how rivers follow the natural contours of the terrain.

Slide 6: This image shows the result of a visibility calculation applied to a user-selected point on the screen (here chosen to lie within a valley).

Slide 7: This image provides what can be thought of as an evaluation of the risk associated with a particular movement path. Here, the visibility calculation used in the previous slide has been applied to a sequence of points along the valley, and the system then color coded the number of times outlying spots were seen. White indicates once, blue indicates 2-3 times, and red indicates more than 3. There is a small area of red in the lower right hand section of the image which suggests a likely spot for placement of an enemy observer.

III. TAPS Output

The collection of slides attached to this appendix give some examples of the terrain queries and display output it is possible to produce with TAPS. We describe the content of those slides below. Please refer to section 5.3.4 for a description of the TAPS screen. Note that only three copies of this report contain a set of these color 35 millimeter slides.

The first few slides show the effect of different overlay modes. The two overlays involved were produced by the following queries:

```
(retrieve-region (nearby deciduous 15))
(retrieve-region (nearby coniferous 15))
```

These correspond to envelopes (within 15 pixels) of all grid cells that are defined in the underlying CATTS data as part of coniferous or deciduous forests.

Slide 1: Nearby-coniferous-15 is shown in solid overlay mode on top of nearby-deciduous-15.

Slide 2: Nearby-deciduous-15 is shown in solid overlay mode on top of Nearby-coniferous-15.

Slide 3: This shows the same two overlays in 'stiple' mode, where only a percentage of the pixels in each overlay are displayed. Note that 3 different types of regions are distinguishable; purely coniferous, purely deciduous, and the regions where both features are present. The latter show up in a tannish color, although no such color was actually displayed (see the following slide). When different color pixels are displayed in such close proximity, the human eye perceives an averaged color.

Slide 4: This presents a closeup of the results of stippling shown in slide 3. Regular patterns of dots are used as a sampling pattern, and those patterns are offset in order to prevent one sampled overlay from completely aligning with any other.

Slide 5: This shows the same two overlays in 'hashed' mode, meaning that a regular hashed line pattern was used to sample each overlay before display.

Slide 6: This shows a display composed of three overlays, with the forested region in solid green on the bottom, the areas with slope between 0 and 15 degrees in the middle (stippled blue), and the urban areas on top (stippled red). The bright green areas are places where the forested pane shows through (i.e., where the slope is outside the 15 degree tolerance and no cities are present), the blue green encodes forested and acceptable slope, and the reddish tint shows location of cities. The query producing the slope data is shown in the GINDB interaction window, and the binary image resulting from that query is shown in the source display.

Slide 7: This slide is the same as slide 6, except that the slope-0-15 pane has been replaced by one representing slope between 0 and 4 degrees. Note that more bright green shows up (fewer forested areas fit within the slope tolerance). The urban areas also separate into bright and dull red; the dull red is in areas where slope data is also present, the brighter red corresponds to areas where the slope is greater than 4 degrees.

Slide 8: In this slide, we have manually selected a portion of the display by drawing on the screen. This defines a 1 bit wide feature mask (shown in the source display) which can be used as an input into further GINDB processing. The user has named this region "area of interest".

Slide 9: This slide shows that information from the display can be sent back into GINDB. Here, the user formulates a query for zones of favorable traversability by intersecting (or negating) the binary images for each of the overlays displayed in slide 7, and the area of interest defined above. Note that the MPI binds the user-supplied names for the overlays to the binary images they encode; in effect, this produces region constants which can be passed into GINDB.

Slide 10: This slide shows streams in the area of interest displayed over that area. The query shown in the interaction pane also demonstrates the use of relational data

within GINDB. It requests all things (symbolic objects) of type stream which are *in* (e.g., which overlap) the area of interest. The distinction between symbolic object and region is important; objects can participate in relations and carry an arbitrary set of properties. Regions represent areas of space which are the values of certain GINDB queries; the system itself carries no additional knowledge about their structure.

Slide 11: This slide shows the bridges over the streams returned above. Five queries (and five overlays) were required in the current syntax, one to examine each of the stream objects in the display. The last of these queries is shown in the interaction pane; it requests the region near any bridge which is over the specified stream. One can imagine this as the beginning of a more complicated terrain analysis scenario in which the user examines potential river crossing sites by applying further questions to the area of interest and the objects it contains.

Slide 12: This is a blow-up of the aggregate display in slide 11. Note that rivers are shown over the bridge locations, which have are presented as enlarged circles.

Slides 13-14: These slides show blow-ups of the system menus which are attached to the different windows. Slide 13 shows operations on the overlays in the current aggregate, and slide 14 shows the manipulations which can be performed on the display window.

Slide 15: This slide is a detail of the system palette. The markers indicate the color and overlay mode that will be attached to the next overlay which the user selects via the mouse.

IV. Information Concerning the TAPS Code

The code requires a Symbolics with eight bit color running release 6.1.

The information in this section assumes that you have loaded the carry-tape on your machine, and that the BCA code is now in a directory named "<machine-name>:>BCA". You will find several files at the top level of this directory as well as several directories.

The directories contain source and compliled code:

- "<machine-name>:>bca:>data>" contains the data files for the CATTS database.
- "<machine-name>:>bca:>gindb>" contains the code for the GINDB module.
- "<machine-name>:>bca:>mpi>" contains the code for the Multiple-Pane Interface.
- "<machine-name>:>bca:>shark>" contains the code for the SHARK window package, which is a utility for the Multiple-Pane Interface.

There are two files which you will have to modify, In the file "<machine-name>:>BCA:>hosts.lisp", change the pathnames in italics in the function below to reflect the directory locations on your host:

```
(defun set-BCA-host (host)
  (fs:set-logical-pathname-host "BCA"
                  :physical-host host
                  :translations
                        '(("MPI;"     ">projects>acorn-bca>mpi>")
                          ("GINDB;"   ">projects>acorn-bca>gindb>")
                          ("DATA;"    ">projects>acorn-bca>data>")
                          ("SHARK;"   ">projects>acorn-bca>shark>")
                          ))
  (let ((inhibit-fdefine-warnings :just-warning))
    (si:set-system-source-file 'mpi   "BCA:MPI;system.lisp")
    (si:set-system-source-file 'gindb "BCA:GINDB;system.lisp")
    (si:set-system-source-file 'sh "BCA:SHARK;system.lisp")
    )
```

```
)
```

In the file "<machine-name>:>BCA:>bca-init.lisp", change the pathnames in the lines indicated in italics to reflect your local host and file locations:

```
(defun init-BCA ()
  (send terminal-io :set-more-p nil)
  (load "a:>projects>acorn-bca>hosts.lisp")
  (set-BCA-host "acorn")
  (make-system 'sh  :compile :noconfirm)
  (make-system 'mpi :compile :noconfirm)
  (make-system 'gindb :compile :noconfirm)
  (mpi:init-mpi)
  (send terminal-io :set-more-p t)
  (pkg-goto 'gindb)
  )
```

Once these changes have been introduced, initialize the system with the following command sequence:

```
(load "<machine-name>:>bca>bca-init.lisp")
(init-BCA)      ; This will take around  5-10 minutes.
```

You are then ready to experiment with TAPS.

IV.1 Running the Demonstration

We have created a canned demonstration which provides a good introduction to TAPS, and is useful for explaining the capabilities of the system. The demonstration is easy to load with the following instructions.

```
(load "<machine-name>:>bca>bca-init.lisp")
(init-BCA)      ; This will take around  5-10 minutes.
<FUNCTION-X>    ; Change mouse back to monochrome screen.
                ; Choose a lisp-listener.
(init-gindb)    ; This will take around 15-20 minutes.
<FUNCTION-X>    ; Change the mouse back to the color screen.
```

You should type your gindb queries to the GINDB-Listener. The sample demonstration script is found in the file "<machine-name>:>BCA:>demo-script.text".

Note that the menu in the upper-lefthand corner of the color screen is non-operative at the current time.